THESE

présentée à

L'UNIVERSITE DE PARIS VI

pour l'obtention du

DOCTORAT DE L'UNIVERSITE

Spécialité: INFORMATIQUE

par Mademoiselle Martine NIVELT
Monsieur Joël GOY

Sujet de la thèse: Structuration et applications d'un simulateur d'automates.

Soutenue le 10/2/1975 devant la Commission d'examen

composée de: Président Monsieur J. BERSTEL

Examinateurs Monsieur C. GIRAULT
Monsieur B. ROBINET

Invité Monsieur LAFARGUE

THESE

présentée à

L'UNIVERSITE DE PARIS VI

pour l'obtention du

DOCTORAT DE L'UNIVERSITE

Spécialité : INFORMATIQUE

par Mademoiselle Martine NIVELT
Monsieur Joël GOY

Sujet de la thèse : Structuration et applications d'un simulateur d'automates.

Soutenue le 10/2/1975 devant la Commission d'examen

composée de : Président Monsieur J. BERSTEL

Examinateurs Monsieur C. GIRAULT
Monsieur B. ROBINET

Invité Monsieur LAFARGUE

UNIV.
LEIDEN
BIBL.

Nous tenons à remercier en premier lieu Monsieur BERSTEL qui a bien voulu présider le jury de cette thèse, ainsi que Monsieur ROBINET qui a accepté d'en faire partie.

Pour l'aide continue et le soutien qu'ils nous ont apporté, nous voudrions exprimer notre plus vive gratitude à Monsieur GIRAULT et à Madame DAQUIN, sans lesquels ce travail n'aurait pu être mené à terme.

Nous remercions également Monsieur LAFARGUE Directeur de l'ESIEA.

Nous remercions la Société CENTRAL INFORMATIQUE et tout spécialement Mademoiselle RUZE qui a assuré la pénible dactylographie de cette thèse.

Nos vifs remerciements vont également à Madame HENNEQUIN et son équipe qui ont assuré l'impression de cette thèse.

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41548119_0013

PLAN DE LA THESE

CHAPITRE I — INTRODUCTION

CHAPITRE II — LES ENTITES DANS SIMAC

- II - 1 L'ALPHABET
- II - 2 LES VARIABLES D'ETAT
- II - 3 LES FILES
- II - 4 LES AUTOMATES
- II - 5 LES CLASSES.

CHAPITRE III — LA STRUCTURATION DU SYSTEME : SIMAC 1-5

- III - 1 HISTORIQUE
- III - 2 LES METHODES DE STRUCTURATION
  - III - 2.1 LA PROGRAMMATION DESCENDANTE
  - III - 2.2 LA MODULARITE
  - III - 2.3 LA PARAMETRISATION
  - III - 2.4 LA REENTRANCE
- III-3 CONCLUSION

CHAPITRE IV — L'ANALYSE FONCTIONNELLE

- IV - 1 IDENTIFICATION DES FONCTIONS
  - IV - 1.1 FONCTIONS PRINCIPALES
  - IV - 1.2 FONCTIONS SECONDAIRES
  - IV - 1.3 FONCTIONS UTILITAIRES
- IV - 2 DEFINITION DES MACHINES FONCTIONNELLES
  - IV - 2.1 MACHINE INITIALISATEUR
  - IV - 2.2 MACHINE MONITEUR
  - IV - 2.3 MACHINE CREATION
  - IV - 2.4 MACHINE MODIFICATION
  - IV - 2.5 MACHINE EXECUTION
  - IV - 2.6 MACHINE OPERATEURS
  - IV - 2.7 MACHINE ARCHIVAGE
  - IV - 2.8 MACHINE ALLOCATEUR
  - IV - 2.9 MACHINE ANALYSE
  - IV - 2.10 MACHINE VISUALISATEUR
  - IV - 2.11 MACHINE UTILITAIRES

CHAPITRE V L'ANALYSE ORGANIQUE

V - 1 IDENTIFICATION DES MODULES

V - 2 SPECIFICATION

V - 2.1 MACHINE INITIALISATEUR

V - 2.2 MACHINE MONITEUR

V - 2.3 MACHINE CREATION

V - 2.4 MACHINE MODIFICATION

V - 2.5 MACHINE EXECUTION

V - 2.6 MACHINE OPERATEURS

V - 2.7 MACHINE ARCHIVAGE

V - 2.8 MACHINE ALLOCATEUR

V - 2.9 MACHINE ANALYSE

V - 2.10 MACHINE VISUALISATEUR

V - 2.11 MACHINE UTILITAIRES

CHAPITRE VI CONCLUSION

ANNEXE 1 BIBLIOGRAPHIE

ANNEXE 2 DICTIONNAIRE

ANNEXE 3 CAMPAGNE D'ESSAIS.

CHAPITRE I

INTRODUCTION

Les machines idéales, telles celles utilisées en théorie des langages (automates, machines de Turing) n'avaient jusqu'à présent été enseignées que de manière conventionnelle. Pourtant, les aspects pratiques liés à l'écriture de leurs règles, à la vérification de leur correction, à leur expérimentation, à leur insertion dans un contexte plus vaste, s'apparentent beaucoup à ceux que l'on rencontre en programmation pratique. Partant, il était tentant d'utiliser un ordinateur comme aide à l'enseignement et de concevoir un sous-système conversationnel dans le cadre d'un système en temps partagé [DAQ2]. Précisément SIMAC, simulateur conversationnel d'automates, se propose d'offrir à l'usager un outil souple et maniable pour la mise en oeuvre de machines idéales [LEN, MAR, HOP]. De telles machines n'avaient pu être l'objet d'un enseignement ou d'une expérimentation faute de systèmes susceptibles de les accepter puis de les exécuter. Contrairement à la plupart des systèmes antérieurs, SIMAC n'oblige pas l'expérimentateur à dénaturer les machines idéales en les transformant au préalable en programmes conventionnels [KNU, CUR].

Une machine d'un type donné est décrite par un ensemble de règles définissant son action sur un ensemble d'entités constituant son contexte ; l'ensemble de ses opérateurs est chargé de manipuler ce contexte. SIMAC possède à cet effet un ensemble d'opérateurs standard. Le nombre des opérateurs et la possibilité de les combiner suffisent pour l'ensemble des machines abstraites faisant l'objet d'un enseignement classique. Pour définir un type de machine, l'usager doit spécifier ses opérateurs, ce qui détermine une forme canonique des règles pour ce type de machine.

SIMAC orienté exclusivement vers l'usage conversationnel, possède un important langage de commande afin de permettre :

. la définition de types d'automates et des formes de règles associées,

. l'introduction des règles,

. la constitution du contexte et l'exécution des automates,

. la visualisation des automates et de leur contexte.

Dans une première étape, nous avons procédé à une campagne d'essais intensifs de SIMAC non pour le mettre au point mais pour améliorer la spécification externe du langage de commande et les commodités d'utilisation conversationnelle (annexe 3).

Pour les besoins de l'enseignement, il est utile de pouvoir éventuellement définir de nouveaux types d'automates ou de données ; il fallait donc que SIMAC soit évolutif, car les extensions sont alors programmées à l'intérieur du système. Afin de pouvoir réaliser commodément de telles modifications de SIMAC, il est indispensable de disposer d'un système particulièrement bien structuré et décrit. Or SIMAC, système opérationnel depuis Septembre 1970, a été constamment étendu. Parce que SIMAC est un objet de recherche et non d'exploitation figée, plusieurs versions parallèles ont subi des évolutions successives incompatibles. De plus se sont encore introduites quelques spécifications dispersées réalisées par des équipes de DEA indépendantes :

Il n'était pas intéressant de se donner systématiquement la peine de restructurer entièrement SIMAC à chaque extension. Ainsi SIMAC, système défini historiquement par des adjonctions successives comportait de nombreuses ramifications parfois assez embrouillées. Ceci avait pour conséquence de rendre sa compréhension de plus en plus malaisée et ainsi de freiner la mise en oeuvre de nouvelles extensions.

Notre travail a précisément été la restructuration complète de SIMAC, l'intégration et la reprogrammation de nombreuses parties. La restructuration d'un ensemble aussi vaste que SIMAC ne peut être envisagée sans redéfinir chacun des objets fondamentaux du système. Bien des propriétés ont en effet été surimposées aux définitions originales des objets ; il a donc fallu étendre les propriétés de certains objets, créer de nouveaux objets puis normaliser les définitions, réexaminer les relations entre objets et enfin assurer la cohérence de l'ensemble.

Nous avons aussi tenté d'introduire dans les faits sinon dans le langage la notion de type de donnée [LIS] . A chaque type a été associée une structure protégée munie d'opérateurs spécialisés pour sa gestion. A l'extérieur du module la structure n'est plus accessible que par ces opérateurs. Il en résulte à la fois une protection des propriétés de la structure manipulée et une transparence des détails d'implantation.

Pour décrire SIMAC, nous n'avons pas essayé d'utiliser effectivement un langage de haut niveau (PASCAL, ALGOL68, ...) afin de ne pas avoir à nous soumettre aux contraintes inhérentes à ce genre de langages. Nous avions en effet le double souci d'une part d'être lisibles d'autre part de pouvoir traiter avec concision notre problème particulier. Nous avons donc défini un langage de spécification, autodocumenté parce qu'assez proche du français et néanmoins muni de structures rigoureuses de contrôle.

Ce travail a été mené dans l'optique d'une programmation descendante telle qu'elle a été définie par Wirth [WIR], Dijkstra [DIJ]. En bref nous avons cherché, comme le préconisait Dijkstra à "rendre le programme si bien structuré que l'effort intellectuel nécessaire à sa compréhension soit proportionnel à sa longueur". La reprogrammation de SIMAC dans un nouveau langage, tout comme une extension éventuelle, seraient maintenant une tâche aisée.

Après cette introduction (Chapitre I), nous présentons les objets du langage ainsi que quelques exemples de manipulation de ces objets (Chapitre II).

Le chapitre suivant (Chapitre III) traite des méthodes utilisées pour l'écriture de SIMAC. Les notions de programmation structurée, de modularité, de paramétrisation sont présentées d'un point de vue général puis appliquées à la structuration du simulateur.

Puis une analyse fonctionnelle dégage les diverses fonctions de SIMAC et nous permet d'associer à chacune d'elles une machine fonctionnelle (Chapitre IV).

Au cours de l'analyse organique (Chapitre V), nous exposons la structuration proprement dite de SIMAC : découpage des machines fonctionnelles en modules et spécification des interfaces entre les différents modules (primitives). Les quelques organigrammes présents ne servent qu'à donner une vue synoptique de certains traitements, car le langage de spécification est suffisant pour décrire avec précision les traitements effectués.

Sont situés en Annexes après la conclusion (Chapitre VI), la bibliographie (Annexe 1), l'index des primitives (Annexe 2) qui, loin d'être un classique tableau de références croisées, a lui-même été structuré de manière à refléter fidèlement la hiérarchie d'imbrication des différents modules, enfin la campagne d'essais (Annexe 3) menée sur SIMAC

En raison de la complexité des mécanismes nécessaires pour simuler des classes variées d'automates, du volume de l'analyse et de l'importance de la programmation due à l'introduction de nouveaux concepts, ce travail a été effectué à deux. Martine NIVELT s'est tout particulièrement penchée sur les méthodes d'organisation modulaire des programmes, sur l'analyse fonctionnelle, la spécificication des objets internes du système et sur les mécanismes d'archivage des entités (y compris les automates).
Joël GOY s'est préoccupé des mécanismes d'exécution des automates, de l'expérimentation des programmes initiaux, de la redéfinition du langage de commandes et des aides à la mise au point.

CHAPITRE II

LES ENTITES DANS SIMAC

Le langage de commande de SIMAC permet de manipuler divers types d'entités. L'ensemble de ces entités constitue l'environnement de l'usager et peut être utilisé par les automates créés par cet usager.

Nous sommes amenés à donner des exemples de "conversations" entre SIMAC et l'usager. Chaque fois qu'il est amené à attendre des informations de l'usager, le système indique par un message approprié la nature des informations demandées plutôt que de laisser l'usager dans l'expectative. Pour permettre de reconstituer la succession des questions et des réponses, nous avons systématiquement souligné ce qui était imprimé par SIMAC.

Les types d'entités actuellement disponibles sont les variables d'état, les files, les automates et les classes. Toutes ces entités sont des structures comportant des chaînes de caractères dont on précise d'abord l'alphabet.

II.1 L'ALPHABET

On note $\mathcal{A}$ l'alphabet composé des symboles disponibles sur les télétypes à l'exclusion de ceux utilisés par le système ou par SIMAC lui-même :

$$\{\ ,\ \mathrm{LF} \rightarrow '\ '\ \}$$

soit alors $\mathcal{A}^*$ l'ensemble des mots (éventuellement vides) sur $\mathcal{A}$.

II-2 LES VARIABLES D'ETAT :

Les seules variables utilisables dans SIMAC 1.5. sont les variables d'état qui prennent pour valeur des mots dans A^* dont la longueur maximum est fixée par l'usager.

* * * * * exemple :

** COM : DEFINIR : VARIABLE = VAR-ETAT (10)

a pour effet de créer une variable d'état dont la longueur est de 10 caractères.

** COM : LIRE
Q0

donne à la variable la chaîne de caractères : Q0 comme contenu.

* * * * *

Les fonctions opérant sur les variables d'état sont les suivantes :

- création d'une variable
- affectation d'un contenu à une variable
- copie d'une variable dans une autre
- édition d'une variable.

II.3 LES FILES :

Les files sont des structures composées d'une suite de caractères de A de longueur maximum fixée par l'usager et d'un pointeur répérant l'emplacement d'une unique tête de lecture-écriture.

* * * * * * exemple :

* * COM : DEFINIR : FILE = B-AUX (50)

a pour effet de créer une file dont la longueur est de 50 caractères.

* * COM : LIRE : B-AUX
C0011011100

donne à la file la chaîne 0....0 comme contenu.

* * COM : POSITIONNER : B-AUX , DEBUT + 3

place la tête de lecture 3 caractères après le début.

* * COM : COPIER : B-ETAFIN = B-AUX

copie la file B-AUX dans l'entité B-ETAFIN précédemment définie.

* * COM : EDITER : B-ETAFIN, COURANT, FIN-1
1101110

édite la partie de la file comprise entre le pointeur courant et le caractère précédant la fin de la file.

* * * * *

Les fonctions opérant sur les files sont les suivantes :

- création d'une file
- affectation d'un contenu à une file
- copie d'une file dans une autre
- positionnement de la tête de lecture
- édition d'une file.

II.4 LES AUTOMATES :

On appelle CONTEXTE d'un automate un ensemble d'entités prises dans l'environnement ; entités auxquelles l'automate est susceptible de s'appliquer.

Il existe des utilisations spécifiques de chaque entité pour lesquelles on a défini des OPERATEURS spécialisés capables d'effectuer toutes les manipulations nécessaires.

Un automate est composé d'un ensemble d'opérateurs. Il est par ailleurs muni d'un ensemble de REGLES, initialement vide, qui décrit le comportement de chacun des opérateurs en fonction du contexte.

On spécifie la FORME DES REGLES d'un automate en choisissant les opérateurs qui le composent . La version actuelle de SIMAC restreint un automate à ne comporter au plus qu'un opérateur de chacun des types suivants :

- ETAT qui s'applique à une variable d'état,
- ENTREE, PILE, BANDE, SORTIE qui s'appliquent à des files.

Un régle comporte :

- un MEMBRE GAUCHE terminé par ->
- un MEMBRE DROIT terminé par 'fin de ligne'

Les deux membres sont des suites de CHAMPS séparés par des " , "

A un opérateur on peut associer un champ dans le membre gauche, un champ dans le membre droit et éventuellement un champ AVANCE dans le membre droit. La possibilité de présence d'un champ est définie par le tableau suivant ;

Opérateur	champ dans mbre gauche	dans le mbre droit	champ AVANCE	type entité
ETAT	x	x		variable d'état
ENTREE	x		x	file
PILE	x	x		file
BANDE	x	x	x	file
SORTIE		x	x	file

* * * * * exemple : création d'un automate à état finis.

* * COM : DEFINIR : AUTOMATE = ETATFIN

* * CHOIX DES OPTIONS :

ETAT : OUI
ENTREE : OUI
AVANCE ENTREE : NON
PILE : NON
BANDE : NON
SORTIE : NON

* * FORME DES REGLES :

ETAT, ENTREE → ETAT

* * * * *

L'automate existe dès qu'il est défini mais avec un ensemble de régles vide ; cet ensemble de règles peut être constitué au fur et à mesure par ajouts successifs il peut ainsi être modifié et édité :

* * * * * exemple :

* * COM : AJOUTEREGLE : ETATFIN

REGLE : Q0 , 1 -> Q1
REGLE : Q0 , 0 -> Q0
REGLE : Q1 , 1 -> Q0
REGLE :

* * COM : EDITER : ETATFIN, REGLE

Q0, 1 -> Q1
Q0, 0 -> Q0
Q1, 1 -> Q0

* * COM : AJOUREGLE : ETATFIN

REGLE : Q1,0 -> Q1

* * * * *

On ne peut activer un automate que s'il dispose d'un contexte complet. On constitue ce contexte en prenant dans l'environnement de entités de types correspondant aux opérateurs figurant dans la définition :

* * * * * exemple :

* * COM : PRENDRE ; ENTREE = B-ETATFIN
* * COM : PRENDRE : ETAT = Q0

* * * * *

Le contexte étant formé, il faut maintenant choisir l'automate que l'on va activer :

* * * * * exemple :

* * COM : FIXER : ETATFIN

* * * * *

Lors d'une exécution : un champ gauche teste des conditions sur l'entité et le membre gauche permet de décider si la règle est APPLICABLE ; un champ gauche ou un champ droit définit des actions sur l'entité, le membre gauche et le membre droit définissent l'APPLICATION de la règle :

* * * * * exemple :

* * COM : PAS APAS

EXECUTION DETERMINISTE : OUI

EDITION CONTEXTE : OUI

EDITION REGLE : OUI

REGLE APPLIQUEE : Q0,1 -> Q1

CONTEXTE :

ETAT : 01

ENTREE : 101110

ENTREE : VIDE

C'EST FINI

* * * * *

Un champ avance définit le déplacemnt de la tête d'une file, en son absence ce déplacement est implicite.

L'ensemble des règles propres à un automate peut être constitué et archivé modifié et édité par les commandes ; les champs d'une règle sont des mots dans A^* à l'exception des champs avance qui sont des entiers dans $\mathbb{N}$

II.5 LES CLASSES :

Il s'agit d'une généralisation de la notion de caractère, grâce à des mécanismes plus élaborés de reconnaissance on arrive alors à réduire considérablement l'écriture des automates.

Une entité de type classe est un ensemble de caractères de $\mathcal{A}$, en particulier il existe des classes standard qui sont :

. ALPHA pour l'ensemble des caractères de $\mathcal{A}$,

. L pour l'ensemble des lettres,

. C pour l'ensemble des chiffres,

. LC pour l'ensemble des lettres et des chiffres.

* * * * * exemple :

** COM : EDITER : C

0 1 2 3 4 5 6 7 8 9

* * * * *

CHAPITRE III

LA STRUCTURATION DU SYSTEME

III.1 Historique

La première version de SIMAC permettait la simulation d'un seul automate ne manipulant à la fois qu'un seul des objets suivants : état, entrée, pile, sortie.

Un premier ensemble d'améliorations a permis d'obtenir les facilités suivantes :

- existence de plusieurs automates indépendants,
- exécution déterministe et/ou en pas à pas des automates, c'est-à-dire règle par règle,
- structuration des règles.

La version résultante SIMAC 0.4 [DAQ 1] est utilisée comme outil d'enseignement et de recherche.

Les améliorations suivantes furent :

- l'introduction des déclarations : on dégagea ainsi la notion de type d'entité précisant uniquement la structure et la valeur et celle d'opérateur d'automate chargé de la manipulation,
- l'introduction de la notion de contexte : ensemble des entités manipulées lors de l'exécution d'un automate,
- l'introduction de l'exécution contrôlée d'un automate : on permet à l'utilisateur de mener à son gré une exécution en combinant : activation déterministe et non déterministe, édition, modification du contexte, poursuite de l'exécution, etc.
- l'introduction d'un nouveau type d'entité : les classes de caractères : généralisation de la notion de caractère,
- l'introduction d'un analyseur lexical facilitant le traitement des commandes.

L'implantation de ces nouvelles extensions s'effectua le plus souvent en parallèle, conduisant ainsi à la mise au point de plusieurs versions intermédiaires : de SIMAC 1.0 à SIMAC 1.4 inclus.

Ainsi, de versions en versions, SIMAC était devenu un programme assez embrouillé ; l'accès aux structures internes étant ainsi rendu malaisé pour le programmeur désireux de définir de nouveaux types d'automates. Or SIMAC est conçu pour offrir la possibilité de travailler avec des machines aussi variées que possible : le programmeur devait pouvoir rapidement et sans trop d'effort introduire les nouvelles entités et les nouveaux opérateurs nécessaires à la mise en oeuvre d'un nouveau type de machine.

Il était donc indispensable de disposer d'un système compréhensible par l'utilisateur et dont l'architecture permettait la programmation d'évolutions : nous avons donc analysé et écrit SIMAC 1.5.

III.2 Les méthodes de structuration utilisées

L'analyse et l'écriture d'un système sont, fondamentalement, la création d'une description complète et précise du système, à partir d'une description du problème à résoudre.

La programmation descendante permet d'obtenir une telle description du système.

III.2 - 1 La programmation descendante [WIR, GOO, MIL]

Dans cette méthode d'analyse, le passage de la description du problème à celle du système se fait de la façon suivante :

1. Un ensemble de tâches est défini par spécification de leurs interfaces. Chaque tâche résoud une partie du problème original ; les interfaces precisent les échanges d'informations entre les tâches.

2. Le comportement interne de chaque tâche est alors analysé : l'interface étant définie, la tâche peut être considérée indépendamment des autres et le principe 1. lui est alors appliqué.

En résumé : la construction de la description consiste en une suite d'étapes de division, à chaque étape une tâche donnée est divisée en un ensemble de sous-tâches.

Toutefois, l'imbrication des tâches peut être très complexe, pourtant les objectifs suivants doivent être atteints :

1. A chaque étape l'analyse doit permettre de se convaincre de la justesse du découpage déjà effectué, sinon les divisions antérieures étaient incorrectes et les étapes précédentes doivent être re-étudiées.

2. Un programme étant souvent modifié, l'analyse doit produire un programme dont la structure est telle que certaines tâches puissent être modifiées sans que le programme entier soit à remanier ; il apparaît donc :

- que les tâches doivent être aussi indépendantes que possible, mais que, les tâches ne pouvant être complètement indépendantes, leurs interactions doivent être clairement présentées et normalisées,

- que le programme doit être compréhensible et donc que le rôle de chaque valeur utilisée dans le calcul doit apparaître clairement et que, de plus, chaque valeur doit pouvoir être modifiée sans problème.

III.2 - 2 La modularité

Les objectifs de la conception modulaire des programmes sont les suivants :

- possibilité de s'assurer de la justesse d'un module indépendemment de son contexte d'utilisation dans un système,
- possibilité de combiner des modules, écrits par des personnes différentes, sans connaissance nécessaire des travaux internes effectués.

L'analyse modulaire d'un programme tend à fournir des modules :

- indépendants de leur contexte d'utilisation
- n'interférant pas les uns avec les autres.

La dépendance et l'interférence des modules peuvent avoir comme causes :

- l'utilisation de références externes au module,
- la non-détection des erreurs aux endroits où elles apparaissent,
- les sauts désordonnés à n'importe quels points du programme.

Un module, répondant à ces exigences, se présentera sous la forme d'une procédure réalisant une tâche :

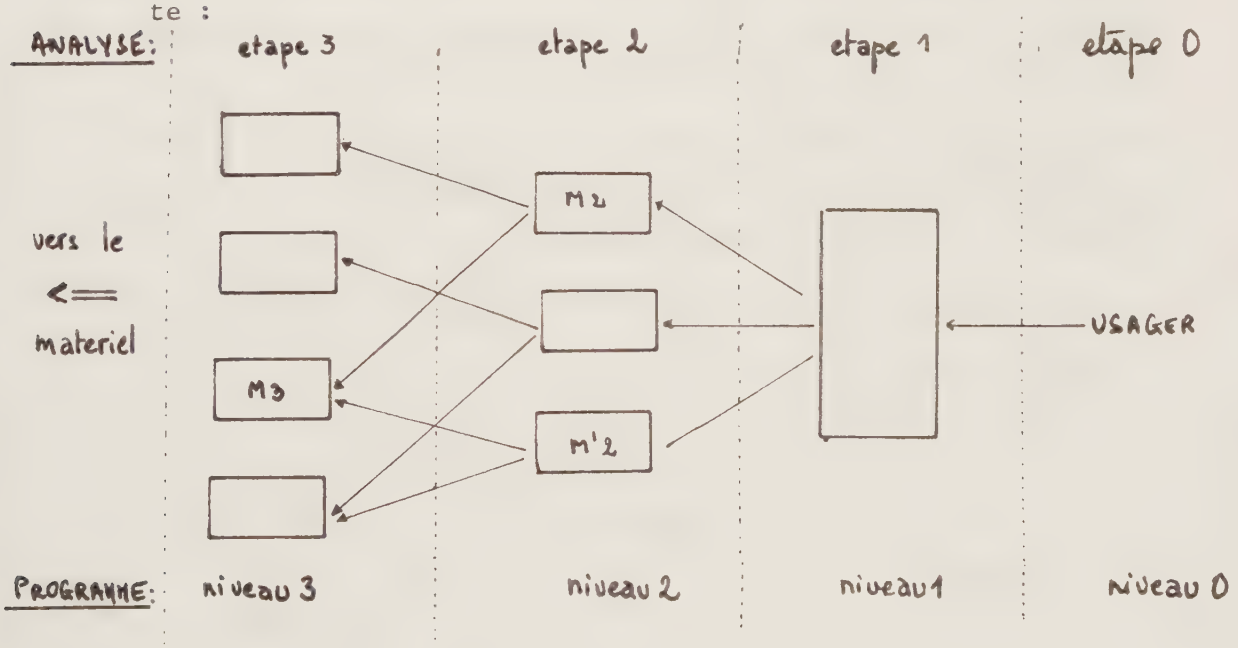
- quand un module est activé, l'ensemble des données nécessaires à son travail est fourni,
- le module effectue alors son travail à partir des données en utilisant ses propres références - ou celles de ses sous-modules sans aucun échange d'information avec l'extérieur. [DEN]

- à chaque étape des tests vérifient la bonne marche du calcul : dans le cas où une erreur est détectée, un code d'erreur est alors positionné et l'activation se termine,
- à la fin de l'activation - fin normale ou sur erreur - on rend au module appelant un code d'erreur et l'ensemble des résultats. Un test sur la réponse lui permettra de savoir si l'ensemble des résultats est utilisable.

Le programme se présente sous la forme d'un ensemble de modules, définis par les tâches qu'ils effectuent, hiérarchisé en niveaux, chaque niveau correspondant à une étape de l'analyse descendante.

Chaque module est donc défini par la visibilité externe des fonctions qu'il remplit : ces fonctions d'activation sont appelées primitives : un module est donc un ensemble de procédures réalisant des primitives.

La structure d'un programme modulaire est donc la suivante :



où :

- M3 est le module de niveau 3 réalisant une tâche apparue dans l'analyse à la 3ème étape lors de la décomposition des tâches définissant les modules M2 et M'2,
- les ⟶ désignent l'émission de la primitive vers le module qui réalise la procédure désirée.

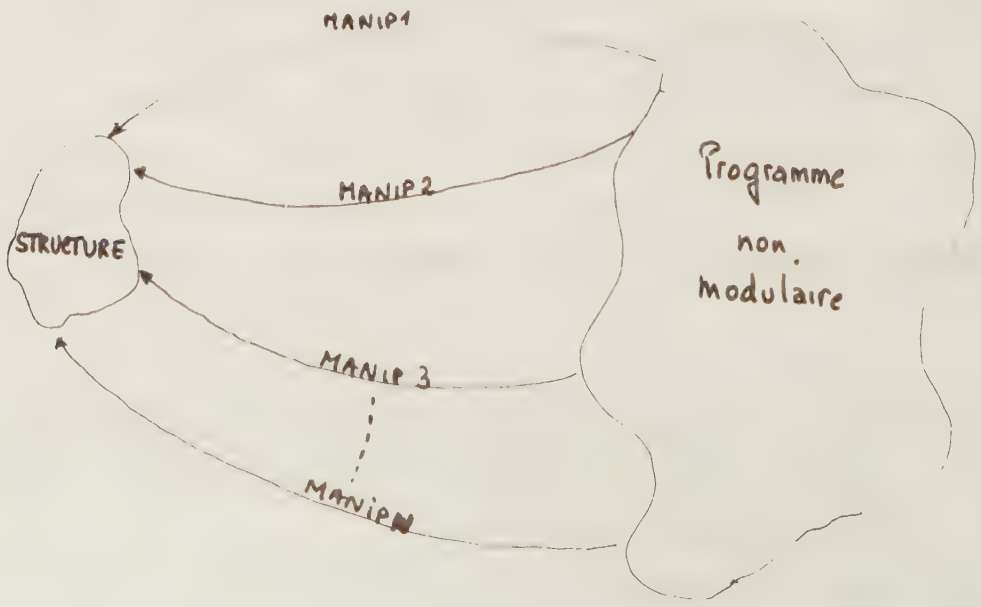
La gestion des structures de données en programmation modulaire s'effectue de la façon suivante :

A chaque structure de données se trouve associé un module dont les primitives réalisent les seules manipulations admises sur la structure : ceci ayant pour effet d'obliger le programmeur à bien définir ces manipulations, puisque tout accès à cette structure ne se fera désormais que par l'activation des primitives [LIS]

Dans le cas d'une programmation non modulaire, les manipulations d'une structure sont dispersées dans le programme : chaque manipulation étant programmée en fonction de la partie de programme où elle figure et non en fonction de la structure manipulée, il peut en résulter les désagréments suivants :

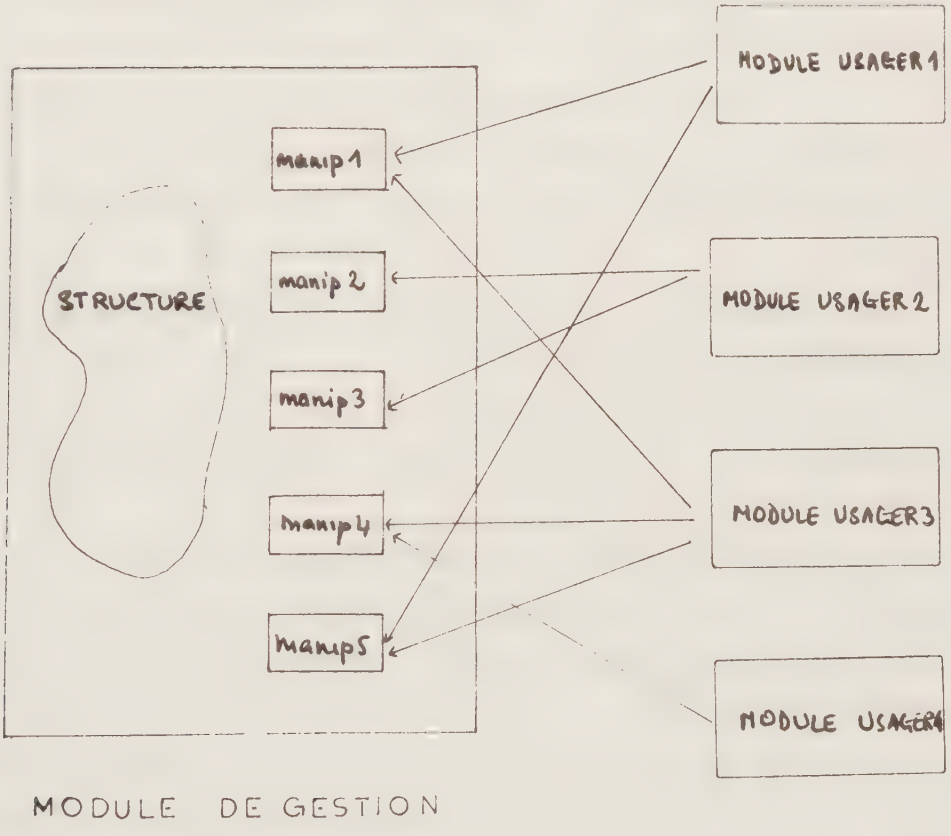
- une manipulation peut être programmée deux fois : le programme n'est donc pas optimal,
- une manipulation peut être erronée : la détection de cette erreur est alors rendue difficile par la dispersion dans le programme des différentes manipulations sur la même structure,
- de plus dans les grands programmes on en arrive à ne plus savoir exactement ce qui se passe au niveau de la structure vu le nombre important des manipulations effectuées.

Cette situation est illustrée dans le schéma suivant :



En associant à chaque structure un module qui regroupe toutes les manipulations nécessaires, on systématise l'accès à la structure : les primitives sont définies en fonction de la structure et représentent l'interface entre le module usager et le module de gestion ; le module usager n'a plus accès à la structure qui lui est transparente.

Le schéma suivant illustre cette méthode de gestion :



III 2 - 3 La paramétrisation

Dans tout programme il existe beaucoup de valeurs arbitraires telles que :

- taille d'un tampon
- nombre d'items d'une table
- numéro d'un item d'une table.

Ces valeurs forment la plus grande partie des constantes d'un programme pourtant il s'agit en fait des variables du niveau 0 d'une analyse descendante : elles seront dans la suite nommées Paramètres du programme.

Si l'on écrit un programme en faisant figurer littéralement ces paramètres, tout changement représente un gros travail matériel qui s'accompagne d'oublis et d'erreurs, et qui devient souvent impraticable.

En associant un nom à chacun des paramètres existant dans un programme on facilite l'adaptation

- du programme à d'autres systèmes,
- du programmeur à son propre programme : cas des modifications
- d'un programmeur quelconque à un programme quelconque.

La paramétrisation consiste lors de l'analyse à utiliser des symboles pour désigner ces valeurs ; l'indépendance nom-valeur ainsi réalisée apporte :

- une grande variabilité de la valeur : pour toute modification seule la ligne du programme qui réalise l'affectation de la valeur est à changer,
- l'invariance de l'algorithme où seul le nom apparaît.

III.2 - 4 La réentrance

Le système sur lequel s'exécute SIMAC doit posséder certaines propriétés afin de permettre une utilisation optimale de SIMAC :

- SIMAC est un système conversationnel : ceci le prédispose à une utilisation sous un système de multiprogrammation permettant de multiplier le nombre des utilisateurs et d'utiliser les temps morts nécessités par les échanges SIMAC-utilisateur : le programme SIMAC sera donc souvent en suspension longue et serait limité s'il ne pouvait être activé que par un seul usager [GIR2]
- SIMAC permet de créer des machines complexes possédant de nombreuses règles et opérant sur un grand nombre d'entités, il apparaît comme nécessaire d'offrir aux utilisateurs des zones dans une mémoire secondaire où les automates et les entités déjà créés sont conservés entre deux séances de travail.

Ainsi SIMAC doit être implanté de façon réentrante sous un système de multiprogrammation muni de mémoires secondaires relativement importantes. Cette implantation permet un traitement en multiprogrammation tout en ne conservant en mémoire qu'une seule copie des instructions. SIMAC possède ainsi un ensemble d'instructions et de constantes invariantes sur toute exécution et un nombre quelconque d'ensembles de variables modifiées par le déroulement du programme.

Cette nécessité de réentrance oblige lors de la conception à séparer le programme invariant et les espaces de données auxquelles il est susceptible de s'appliquer. Il en résulte une meilleure lisibilité, une plus grande sécurité, et une mise au point plus aisée.

III-3 Conclusion : SIMAC 1.5

Le chapitre précédent a montré les avantages d'une analyse descendante conduisant à un programme modulaire et paramétré.

SIMAC ainsi analysé sera donc un programme cohérent, compréhensible et évolutif.

CHAPITRE IV

ANALYSE FONCTIONNELLE

Cette analyse définit les fonctions logiques générales nécessaires au fonctionnement de SIMAC, une fois ces fonctions indentifiées on leur associe des machines qui les réalisent.

IV-1. IDENTIFICATIONS DES FONCTIONS :

IV-1.1. LES FONCTIONS PRINCIPALES définissent le fonctionnement des entités crées par SIMAC. On distingue :

- les fonctions de manipulation des entités :
  - . fonctions de CREATION des entités,
  - . fonctions de MODIFICATION des entités,
  - . fonctions d'EXECUTION des entités,
- les fonctions liées aux OPERATEURS associés aux entités
- les fonctions liées à l'ARCHIVAGE des structures de données utilisées pour représenter les entités, c'est-à-dire les fonctions d'accès aux structures et de manipulation de ces structures.

IV-1.2. LES FONCTIONS SECONDAIRES sont nécessaires au fonctionnement du simulateur. On distingue :

- le MONITEUR réalisant l'enchainement des fonctions.
- le VISUALISATION, c'est-à-dire l'édition des entités ou des résultats d'une exécution.
- l'ANALYSE, c'est-à-dire la lecture des commandes de l'usager.
- l'INITIALISATEUR.
- l'ALLOCATEUR.

IV-1.3. LA FONCTION UTILITAIRE regroupe les programmes de service.

IV-2. DEFINITION DES MACHINES FONCTIONNELLES.

- LA MACHINE INITIALISATEUR
- LA MACHINE MONITEUR
- LA MACHINE CREATION
- LA MACHINE MODIFICATION
- LA MACHINE EXECUTION
- LA MACHINE OPERATEURS
- LA MACHINE ARCHIVAGE
- LA MACHINE ALLOCATEUR
- LA MACHINE ANALYSE
- LA MACHINE VISUALISATEUR
- LA MACHINE UTILITAIRES

IV-2.1. LA MACHINE INITIALISATEUR.

ANALYSE : cette machine réalise l'interface avec le système.

SPECIFICATION : la machine initialisateur :

. activée par le système initialise les autres machines puis active le moniteur

. activée par le moniteur re-initialise les autres machines puis active le moniteur.

IV-2.2. LA MACHINE MONITEUR

ANALYSE : cette machine réalise la fonction d'enchainement. Elle pilote le fonctionnement des autres machines du simulateur grâce aux commandes de l'usager.

Ces commandes ont toutes le format suivant :

x x COM : < commande > : < liste de paramètres >

SPECIFICATION : la machine Moniteur est activée en premier, elle lit la commande puis active la machine correspondante. Elle signale les erreurs (commande inconnue) et est activée en cas d'erreur survenant dans les autres machines ou en fin de travail de la machine activée.

COMMANDES : la liste des commandes reconnues est la suivante :

Machine	Commande	liste de paramètres	Effet
INITIALISATEUR	DEBUTER	Ø	remet à zéro la zone usager
CREATION	DEFINIR	TYPE,NOM,ENTIER	crée une entité
	DETRUIRE	NOM	détruit une entité
MODIFICATION	AJOUTEREGLE	NOM,LISTE DE REGLES	ajoute des règles à un automate
	OTEREGLE	NOM,LISTE DE REGLES	ote des règles à un automate
	AJOUTERCAR	NOM,LISTE DE CARACT.	ajoute des caractères à une classe
	OTERCAR	NOM,LISTE DE CARACT.	ote des caractères à une classe
	LIRE	NOM,POSITION	donne un contenu à une variable ou à une file.
	POSITIONNER	NOM,POSITION	place la tête de lecture sur une file
	COPIER	NOM,NOM	copie une entité dans une autre
EXECUTION	FIXER	NOM,	précise l'automate que l'on va activer
	PRENDRE	NOM,NOM	spécifie le contexte d'activation
	PASAPAS	Ø	)
	ACTIVER	Ø	) active un automate
	CONTINUER	Ø	)
VISUALISATEUR	LISTER	TYPE	liste le nom de toutes les entités
	REGLES	Ø	liste les règles appliquées lors de la dernière exécution.
	EDITER	NOM,REFERENCE	édite l'entitée demandée

IV-2.3 LA MACHINE CREATION :

ANALYSE : la machine CREATION assure la création ainsi que la destruction des entités dans SIMAC.

SPECIFICATION : elle est activée par le moniteur sur commande :

- elle crée une entité : le NOM et le TYPE de l'entité doivent être précisés par l'usager ; le type de l'entité détermine sa structure qui est alors créée et initialisée à vide ;

- elle détruit une entité de NOM donné ;
en cas d'erreur le moniteur est activé après restauration des structures manipulées dans leur état initial et édition d'un diagnostic.

IV-2.4.LA MACHINE MODIFICATION

ANALYSE : la machine MODIFICATION se charge de modifier le contenu des entités.

SPECIFICATION : elle est activée par le moniteur sur commande, la commande doit préciser le NOM de l'entité ;
les modifications se font en fonction de la structure de l'entité (structure précisée par son TYPE) ;
en cas d'erreur le moniteur est activé après restauration des structures et édition d'un diagnostic.

IV-2.5 LA MACHINE EXECUTION

ANALYSE : la machine EXECUTION a pour fonction d'activer un automate. Un automate peut-être activé selon plusieurs MODES déterminés par les OPTIONS précisés par l'usager.

Les options suivantes sont à la dispositon de l'usager :

- l'option DETERMINISTE : qui consiste à chercher parmi ses règles archivées la première qui soit applicable, à modifier en fonction de cette règle les valeurs des entités composant le contexte, puis à recommencer jusqu'à ce qu'il n'y ait plus de règle applicable
- l'option NON-DETERMINISTE : dans ce cas il faut effectuer des essais successivement avec toutes les règles applicables. Si il n'y a plus de règles applicables le mécanisme suivant est utilisé :
  - . désapplication de la règle
  - . restauration du contexte
  - . recherche parmi les autres règles archivées d'une règle applicable.

  le mécanisme est itéré tant qu'une nouvelle règle applicable n'a pas été trouvée.

Afin d'éviter un stockage des valeurs successives du contexte, on retient les automates non déterministes à ne posséder que des règles dont l'application soit réversible c'est-à-dire telles que la restauration du contexte soit possible uniquement par la consultation des champs des règles qui ont été successivement appliquées.

.../

- l'option CONTROLE : permet à l'usager de fixer lui-même le nombre de règles à appliquer ou à désappliquer

- l'option PASAPAS : permet à chaque application ou désapplication d'une règle de visualiser la règle et le contexte de l'automate.

SPECIFICATION : activée par le moniteur sur commande :

- elle rend COURANT un automate de NOM donné c'est-à-dire choisit dans l'environnement de l'usager l'automate que l'on va activer

- elle constitue un CONTEXTE à cet automate : on constitue un contexte en prenant dans l'environnement de l'usager des entités de type correspondant aux opérateurs figurant dans l'automate.

Il existe ainsi généralement à un instant donné un automate COURANT et un contexte COURANT et on peut changer l'automate courant sans changer le contexte courant et réciproquement.

- elle active cet automate selon le mode d'exécution choisi par l'usager lors d'un dialogue avec SIMAC ; on distingue :

  * les exécutions non conversationnelles : où, une fois le mode choisi l'usager n'est plus consulté : c'est l'option 'DETERMINISTE' cette option se combine avec l'option 'PASAPAS' pour donner :

- l'exécution déterministe sans pas à pas

- l'exécution déterministe avec pas à pas

* les exécutions conversationnelles : où l'usager est consulté en cours d'exécution, il existe deux options :

    A. l'option NON-DETERMINISTE qui combinée avec l'option PASAPAS donne :

        - l'exécution non déterministe sans pas à pas
        - l'exécution non déterministe avec pas à pas

    B. l'option CONTROLE qui combinée avec l'option PASAPAS donne :

        - l'exécution controle sans pas à pas
        - l'exécution controle avec pas à pas

le pas à pas constituant une commodité qui ne change en rien le déroulement de l'exécution, il y a donc trois MODES d'exécution des automates qui sont :

1. le mode DETERMINISTE-SANS CONTROLE
2. le mode NON DETERMINISTE-SANS CONTROLE
3. le mode avec CONTROLE.

IV-2.6. LA MACHINE OPERATEURS

ANALYSE : cette machine gère la constitution et la modification du du contexte courant.

SPECIFICATION : activée par la machine exécution, la machine OPERATEURS

- rend courant les opérateursdont on lui donne le nom
- assure les manipulations effectuées sur les éléments du contexte lors d'une exécution.

IV-2.7. LA MACHINE ARCHIVAGE

ANALYSE : cette machine réalise les fonctions d'accès aux structures utilisées pour représenter les entités, et de manipulation de ces structures.

SPECIFICATION : activée par les machines de manipulation des entités et par le visualisateur, elle exécute sur les structures les traitements demandés : création, recherche et destruction d'une entité.

IV-2.8.LA MACHINE ALLOCATEUR

ANALYSE : cette machine gère l'espace alloué à l'usager pour son travail.

SPECIFICATION : activée lors de la création d'une structure la machine ALLOCATEUR alloue l'espace nécessaire à sa création.

Il n'existe pas,à proprement parler, de primitive de désallocation dans SIMAC l'espace correspondant aux entités détruites est pour le moment perdu.

Il existe toutefois une récupération simple sans tassage, ni réorganisation des structures très complexes manipulées par SIMAC

IV-2.9.LA MACHINE ANALYSE

ANALYSE : cette machine analyse les commandes et les réponses de l'usager.

SPECIFICATION : activée par les machines de manipulation des entités et par le visualisateur la machine ANALYSE :

- réalise l'analyse ajutaxique des commandes lues.
- gère le tampon d'entrée des messages de l'usager.

CHAPITRE 5

ANALYSE ORGANIQUE

Cette analyse a pour but de résoudre les problèmes inhérents à la réalisation du simulateur :

- découpage en modules des machines complexes
- spécification des interfaces entre modules.

V-1. IDENTIFICATION DES MODULES

Les modules correspondent à des morceaux de machine réalisables séparément. Ils peuvent éventuellement coïncider avec une machine si celle-ci est simple.

Les machines :

INITIALISATEUR

MONITEUR

ALLOCATEUR

CREATION

MODIFICATION

UTILITAIRES

ne contiennent qu'un seul module, les machines :

EXECUTION

OPERATEURS

ARCHIVAGE

ANALYSE

VISUALISATEUR

sont au contraire décomposables :

. LA MACHINE EXECUTION

On y distingue 4 modules :

1) le module FIXER qui précise l'automate sur lequel on va travailler
2) le module CONTEXTE qui spécifie le contexte de l'automate choisi
3) le module ACTIVATION qui active cet automate. Ce dernier module contient trois sous-modules :

. le ss-module CHOIX qui précise le mode d'exécution choisi

. le ss-module EXECUTEURS qui regroupe les 3 exécuteurs correspondant aux 3 modes d'exécution

. le ss-module FONCTION D'EXECUTION qui regroupe les fonctions qui apparaissent dans l'exécution d'un automate.

4) le module PILEXEC qui gère la pile d'exécution

. LA MACHINE OPERATEUR :

On y distingue 2 modules :

1) le module TABLE DES OPERATEURS COURANTS qui gère la table des opérateurs courants

2) le module TRAITEMENT DES OPERATEURS qui regroupe les procédures associées aux opérateurs.

. LA MACHINE ARCHIVAGE :

On y distingue 2 modules :

1) le module IDENTIFICATION qui gère la structure associée aux identifications

2) le module CONTENU qui gère la structure associée aux contenus des entités

le module contient 3 sous-modules

. le ss-module GESTION DES AUTOMATES

. le ss-module GESTION DES CLASSES

. le ss-module GESTION DES FILES ET DES VARIABLES

. LA MACHINE ANALYSE :

On y distingue 2 modules :

1) le module ANALYSEUR SYNTAXIQUE qui reconnait les éléments du vocabulaire des commandes.
2) le module FONCTIONS DE LECTURE qui gère le tampon d'entrée.

. LA MACHINE VISUALISATEUR :

On y distingue 2 modules :

1) le module DEMANDE DE VISUALISATION qui réalise les éditions demandées.
2) le module FONCTIONS DE VISUALISATION qui gère le tampon de sortie

SPECIFICATIONS :

On donne ici pour chaque module :

1) l'objet du module c'est-à-dire sa fonction,
2) les structures manipulées par ce module,
3) son interface avec les autres modules, c'est-à-dire les primitives de ce module.

L'activation d'une primitive sera symbolisée par l'appel d'une fonction FORTRAN, accompagnée de tous ses paramètres, bien que la passation des arguments ne se fasse pas ainsi. On spécifie ensuite ;

1) son objet,
2) ses arguments, d'activation et arguments résultat,
3) son code REPONSE.

On obtient ainsi la forme :

REP = < nom de primitive > (<liste des paramètres>).

-2.1.LA MACHINE INITIALISATEUR

OBJET : cette machine initialise les autres machines puis active le moniteur.
Quand une re-initialisation est voulue par l'usager, le moniteur active DEBUTER.

PRIMITIVES :

REP = DEBUTER
Pas d'arguments.
OBJET : Initialise avant travail sur SIMAC :

. initialise l'allocateur en émettant :

INITALLOC

. initialise archivage en émettant :

INITLIST

. émet : IMPRIME (SIMAC 1.5)

. émet : ACTIMONI vers le moniteur

Pas de réponse.

REP = REPRISE
Pas d'arguments.
OBJET : permet de retrouver un travail déjà commencé dans l'état où il avait été laissé à l'issue d'une commande ABANDON :

. émet : IMPRIME (REPRISE SIMAC 1.5)

. active le moniteur en émettant : ACTIMONI

Pas de réponse.

V-22 LA MACHINE MONITEUR

OBJET : pilote le fonctionnement des autres machines du simulateur grâce aux commandes de l'usager.

PRIMITIVE :

REP = ACTIMON1

Pas d'argument.

OBJET : traite la commande de l'usager :

1) . demande une commande en émettant :
LECT (COM :) si REP = 0 alors retourne en 1)

. traite la réponse de l'usager en émettant :
CHNOM (OBLIG, NOM) si REP = 2 alors retourne en 1)

. vérifie que la commande lue est correcte :
TABLE (NOM,TCOM,REFITEM) si REP = 1 alors retourne en 1)

. émet la primitive correspondant à la commande.

. retourne en 1)

Pas de code Réponse.

NOTA : si par souci de clarté on a choisi pour les primitives émises, dont la liste suit, des noms en rapport avec les commandes, il n'y a pas identité entre la commande et la fonction FORTRAN qui symbolise l'appel de la primitive exécutant cette commande.

LISTE DES PRIMITIVES EMISES

. vers la machine INITIALISATEUR :

DEBUTER

. vers la machine CREATION :

DEFINIR

DETRUIRE

. vers la machine MODIFICATION :

LIRE

POSITIONNER

COPIER

AJOUTEREGLE

OTEREGLE

AJOUTECAR

OTERCAR

. vers la machine EXECUTION :

FIXER

PRENDRE

PASAPAS

ACTIVER

CONTINUER

. vers la machine VISUALISATEUR :

LISTER

REGLES

EDITER

V-23. LA MACHINE CREATION

OBJET : assurer la création et la destruction des entités dans SIMAC.

PRIMITIVES :

REP = DEFINIR

Pas d'ARGUMENT

OBJET : crée une entité en traitant la commande :

DEFINIR : TYPE = NOM / NOM (ENTIER)

1) analyse la commande :

. recherche un TYPE d'entité, obligatoire en émettant
CHTYPE (OBLIG,ENTITE,PARAM,TYP)
. si REP = - 2 termine.

. recherche un NOM,obligatoire, en émettant :
CHNOM (OBLIG, NOM)
. si REP = - 2 termine.

. recherche un ENTIER, facultatif, en émettant :
CHENTI (NOBLIG, NBCENT)

2) crée l'identificateur :

. recherche si un identificateur porte déjà le nom donné :
CHLIST (NOM)
. si REP = 0 (nom déjà attribué)
. vérifie que l'item n'est pas une classe standard en émettant :
CLASTAND (REPONSE)
. si REPONSE = OUI
. émet IMPRIME (INTERDIT POUR CLASSE STANDARD)
. termine
. demande si une redéfinition est voulue
OUINON (REDEFINIR, REPONSE)

. si REP = 1 termine
. si REPONSE = NON termine
. si REPONSE = OUI détruit l'item
préexistant en émettant
DETLIST

. insère en liste le nouvel identificateur en émettant
CRELIST (NOM, PARAM, TYP)
. si REP = 1 termine

3) crée l'entité : en émettant
CREENT (TYP, NBLENT, REFSTRUC)
. si REP < 0 termine

4) place la référence du descriptif de l'entité dans
l'identificateur en émettant :
LIDESTRUC (REFSTRUC)

5) émet le message :
IMPRIME (DEFINI)

6) termine

REPONSE : REP = 0 fin normale
REP < 0 fin sur erreur.

REP = DETRUIRE

Pas d'ARGUMENT

OBJET : détruit une entité en traitant la commande :
DETRUIRE : NOM

1) analyse la commande : recherche un NOM obligatoire
en émettant :
CHNOM (OBLIG, NOM) si REP = 2 termine

2) vérifie qu'une entité porte bien le nom, en
émettant :
CHLIST (NOM)
. si REP < 0 : (nom inconnu)
. émet IMPRIME (ENTITE INCONNUE)
. termine

3) si l'entité à détruire est courante - appartient
au contexte courant - détruit l'entité courante
en émettant :
DETCO

4) détruit l'entité en émettant :
DETLIST

5) émet IMPRIME (DETRUIT)

6) termine

REPONSE :

REP = 0 fin normale
REP < 0 fin sur erreur.

2.4. LA MACHINE MODIFICATION

OBJET : se charge de modifier le contenu des entités

PRIMITIVES :

REP = AJOUTEREGLE
Pas d'arguments.

OBJET : ajoute des règles à un automate en traitant la commande :

** COM : AJOUTEREGLE : < NOM >
REGLE : <règle>
REGLE :

1) analyse la commande : recherche un NOM, nom obligatoire
   en émettant : CHNOM (NOBLIG, ZNOM)

2) traite le NOM :
   . si REP > 0 (nom indiqué)
      . recherche si un identificateur porte déjà le nom
        en émettant : CHLIST (ZNOM)
         . si REP = 0 (déjà attribué) vérifie que l'item
           est bien un automate en émettant :
                AUTOMATE (REPONSE)
         . si pas automate : . émet :
                IMPRIME (ENTITE INCONNUE)
                . termine

   . vérifie qu'il existe un automate courant : émet :
      COURANT (REFIDENT, REFSTRUC)
         . si REP = - 1 :
            . émet : IMPRIME (PAS D'AUTOMATE)
            . termine.

3) traite chacune des règles :

   a). initialise avant traitement : en émettant :
                INITREGL (REFSTRUC)

   . lit une règle : en émettant : LIREGL (REFALLOC)
      . si REP = 1 termine

. recherche si cette règle existe déjà : en émettant:
CHREGLE (PTPILE, PTPREC, NUMCP, PTAUT)

si REP > 0 . émet IMPRIME (REGLE DEJA EXISTANTE)
. termine

. ajoute la règle lue aux règles existantes en émettant:
AJOUREGLE (REFALLOC, PTPILE, PTPREC, NUMCP)

. retour en a)

REPONSE :

REP = 0 : fin normale

REP < 0 : fin sur erreur.

REP = OTEREGLE

Pas d'arguments

OBJET : ote des règles à un automate en traitant la commande :

* * COM : OTEREGLE : NOM
REGLE : règle
REGLE :

1) Analyse la commande : recherche un NOM, non obligatoire,
en émettant : CHNOM (NOBLIG, ZNOM)
2) Traite le NOM :
. si REP > 0 (nom indiqué)
. recherche si un identificateur porte déjà ce nom
en émettant : CHLIST (ZNOM)
. si REP = 0 (déjà attribué) vérifie que l'item
est bien un automate en émettant :
AUTOMATE (REPONSE)
si pas automate : . émet :
IMPRIME (ENTITE INCONNUE)
. termine

. vérifie qu'il existe un automate courant : émet :

COURANT (REFIDENT, REFSTRUC)

. si REP = - 1 :

. émet : IMPRIME (PAS D'AUTOMATE)

. termine

3) Traite chacune des règles :

a) . initialise avant traitement : en émettant :

INITREGL (REFSTRUC)

. lit une règle : en émettant :

LIREGL (REFALLOC) si REP = 1 termine

. recherche si cette règle existe déjà :

en émettant : CHREGLE (PTPILE, PTPREC, NUMCP,PTAUT)

si REP = 0 . émet IMPRIME (REGLE INEXISTANTE)

. termine

. ote la règle lue en émettant :

OTREGLE (PTAUT, NUMCP)

. retourne en a)

REPONSE

REP= 0 : fin normale

REP < 0 : fin sur erreur.

REP = AJOUTECAR

Pas d'arguments

OBJET : ajoute des caractères à une classe en traitant la commande :
AJOUTECAR :< NOM >
<liste de caractères>

1) analyse la commande :
. recherche un nom, obligatoire, en émettant :
CHNOM (OBLIG, NOM)
si REP = -2: . émet IMPRIME (ENTITE INCONNUE)
. termine

2) traite le nom :
. recherche si un identificateur porte déjà ce nom en émettant :
CHLIST (NOM)
si REP = -1: . émet IMPRIME (ENTITE INCONNUE)
. termine

. vérifie que l'item courant est une classe, pas standard : émet :
CLASTAND (REPONSE)
si REP = 0 :.si REPONSE = OUI :
. émet IMPRIME (INTERDIT POUR CLASSE STANDARD)
.si REPONSE = NON :
. émet IMPRIME (INCOMPATIBLE)
.termine.

3) traite la liste de caractères :

a). lit la liste des caractères en émettant :
LECT (BLANC) si REP = 0 termine.

b). extrait un caractère de la liste : émet :
CARTAMP (CAR) si REP = -1 retourne en a)

. recherche si le caractère appartient déjà à la classe :

. recherche l'adresse de la classe en émettant :
COURANT (REFIDENT, REFSTRUC)

. émet : CHCAR (REFSTRUC, CAR, NOMOT, RESTE, CARES)

. si le caractère existe déjà : CARES ≠ 0 alors
émet : IMPRIME (<CAR> EXISTE)
sinon : ajoute le caractère en émettant :
AJCAR (REFSTRUC, NOMOT, RESTE)

. retourne en b)

4) termine.

REPONSE :

REP = 0 si fin normale

REP = -1 si erreur.

REP = OTERCAR

Pas d'arguments

OBJET : ôte des caractères à une classe en traitant la commande :
OTERCAR : <NOM>
<liste de caractères>

1) analyse la commande :
. recherche un nom, obligatoire, en émettant :
CHNOM (OBLIG, ZNOM)
si REP = -2 : . émet IMPRIME (ENTITE INCONNUE)
. termine.

2) traite le nom :
. recherche si un identificateur porte déjà ce nom en émettant :
CHLIST (ZNOM)
si REP = -1 : . émet IMPRIME (ENTITE INCONNUE)
. termine.
. vérifie que l'item courant est une classe, pas standard : émet :
CLASTAND (REPONSE)
si REP = 0 : . si REPONSE = OUI :
émet IMPRIME (INTERDIT POUR CLASSE STANDARD)
. si REPONSE = NON :
émet IMPRIME (INCOMPATIBLE)
. termine.

3) traite la liste de caractères :
a). lit la liste des caractères en émettant :
LECT (BLANC) si REP = 0 termine.
b). extrait un caractère de la liste : émet :
CARTAMP (CAR) si REP = -1 retourne en a)

. recherche si le caractère appartient déjà à la classe :

. recherche l'adresse de la classe en émettant :
COURANT (REFIDENT, REFSTRUC)

. émet : CHCAR (REFSTRUC, CAR, NOMOT, RESTE, CARES)

. si le caractère n'existe pas : CARES = 0 alors
émet : IMPRIME (<CAR> N'EXISTE PAS)
sinon : ôte le caractère en émettant :
OTCAR (REFSTRUC, NOMOT, RESTE)

. retourne en b)

4) termine.

REPONSE :

REP = 0 si fin normale

REP = -1 si erreur.

REP = LIRE

Pas d'argument.

OBJET : donne un contenu à une entité de type variable ou file
en traitant la commande :

LIRE :<NOM>, <POSITION>

1) analyse la commande :
. recherche un nom obligatoire en émettant :
CHNOM (OBLIG, ZNOM)
si REP = -2 termine.

. recherche une position :

. REFDESC pointe sur DESCPOS
. émet : CHPOS (REFDESC, TERM)
si REP = 0 :option par défaut:
. RES (REFDESC) : = CODEBUT
. SIGN (REFDESC): = +
. INC (REFDESC) : = 0

2) traite le nom : recherche si un identificateur porte déjà ce nom en émettant :
CHLIST(ZNOM) si REP = 1 alors termine.

3) donne un contenu à l'entité en émettant:
ECRIENT (REFDESC)

4) termine

REPONSE : REP = 0 fin normale
REP = -1 si erreur.

REP : POSITIONNER

Pas d'arguments

OBJET : place le moniteur courant sur une file en traitant la commande :
POSITIONNER : <NOM>, <POSITION>

1) analyse la commande :
   . recherche un nom obligatoire en émettant :
     CHNOM (OBLIG, ZNOM)
     si REP = -2 termine
   . recherche une position :
     . REFDESC pointe DESCPOS
     . émet : CHPOS (REFDESC, TERM)
       si REP = 0 : option par défaut :
       . RES (REFDESC) : = CODEBUT
       . SIGN (REFDESC): = +
       . INC (REFDESC) : = 0

2) traite le nom : recherche si un identificateur porte déjà ce nom en émettant :
   CHLIST (ZNOM) si REP = -1 alors termine.

3) positionne le pointeur courant :
   . recherche l'adresse du descriptif de la file en émettant :
     COURANT (REFIDENT, REFSTRUC)
   . positionne :
     POSTFIL (REFDESC, REFSTRUC)

4) termine

REPONSE : REP = 0 fin normale
REP = 1 si erreur

REP = COPIER

Pas d'arguments

OBJET : copie une entité dans une autre en traitant la commande :
COPIER : <NOM 1> =<NOM Z>

1) analyse la commande et traite les noms :

a). recherche un nom, obligatoire, en émettant :
CHNOM (OBLIG, NOM 1)
si REP = -2 : . émet LECT (NOM 1 :)
. retourne en a)

. recherche si un identificateur porte déjà ce nom, en émettant :
CHLIST (NOM 1) si REP = -1 . émet IMPRIME (ENTITE INCONNUE.)
. termine

b) recherche un nom, obligatoire, en émettant :
CHNOM (OBLIG, NOM Z) si REP = -2 : . émet LECT (NOM Z:)
. retourne en b)

. recherche si un identificateur porte déjà ce nom, en émettant :
CHLIST (NOM Z) si REP = -1 : . émet IMPRIME (ENTITE INCONNUE.)
. termine

2) copie les entités : en émettant :
COPENT (NOM 1) si REP = -1 termine

3) termine

REPONSE :
REP = 0 si fin normale
REP =-1 si erreur.

2.5.LA MACHINE EXECUTION

OBJET : activer un automate.

MODULES .

MODULE FIXER

OBJET : précise l'automate que l'on va activer.

PRIMITIVE :

REP = FIXER

Pas d'arguments

OBJET : prend pour courant un automate en traitant la commande :

FIXER : <NOM >

1) analyse la commande : recherche un nom, obligatoire, en émettant :
   CHNOM (OBLIG, ZNOM) si REP = -2 : . émet : LECT (NOM :)
   . retourne en 1)

2) traite le nom : recherche si un identificateur porte déjà ce nom en émettant :
   CHLIST (ZNOM) si REP = -1 : . émet IMPRIME (ENTITE INCONNUE)
   . termine

3) rend courant l'automate :
   . recherche de l'automate courant en émettant :
   COURANT (REFITEM, REFSTRUC)
   . TYP : = CODAUTO
   . émet COUROPE (TYP, REFITEM) si REP = -1 termine

4) initialise la pile d'execution en émettant :
   INITPILE

5) termine

REPONSE :

REP = 0 si fin normale

REP =-1 si erreur

MODULE CONTEXTE

OBJET : spécifie le contexte de l'automate courant.

PRIMITIVE :

REP = PRENDRE

Pas d'arguments

OBJET : prend pour opérateur courant de l'automate courant l'entité dont on lui donne le nom en traitant la commande :

PRENDRE : <NOM d'opérateur> =<NOM d'entité>

1) analyse la commande et traite les noms

. recherche un nom d'opérateur, obligatoire, en émettant :
CHTYPE (OBLIG, CONTEXTE, PARAM, TYP)

si REP = -1 : . émet IMPRIME (OPERATEUR INCORRECT)
. termine

. recherche un NOM, obligatoire, en émettant :
CHNOM (OBLIG, ZNOM)

si REP = -2 : . émet IMPRIME (NOM ABSENT)
. termine

. recherche si un identificateur porte déjà ce nom en émettant :
CHLIST (ZNOM)

si REP = -1 : . émet IMPRIME (ENTITE INCONNUE)
. termine

2) rend courant :

. calcule l'adresse de l'identificateur : en émettant :
COURANT (REFITEM,REFSTRUC)

. rend courant : émet :
COUROPE (TYP, REFITEM) si REP = -1 termine

3) termine

REPONSE : REP = 0 si fin normale
REP = 1 si erreur

MODULE ACTIVATION

OBJET : choix du mode d'exécution et activation de l'automate courant.

SOUS-MODULES :

sous-module : CHOIX

OBJET : précise le mode d'exécution choisi

PRIMITIVES :

REP = PASAPAS

Pas d'arguments

OBJET : traite la commande :

PASAPAS

par laquelle l'usager demande l'activation d'un automate :

. vérifie si l'exécution est possible :

. vérifie qu'il existe bien un automate courant :

si COAUTO = 0 alors : . émet IMPRIME (PAS D'AUTOMATE)

. termine

. vérifie le contexte en émettant :

VERI si REP = -1 termine.

. vide la pile d'exécution : en émettant :

INITPILE

. positionne la cellule CONTPAS à 2 pour entrée future en CONTINUER

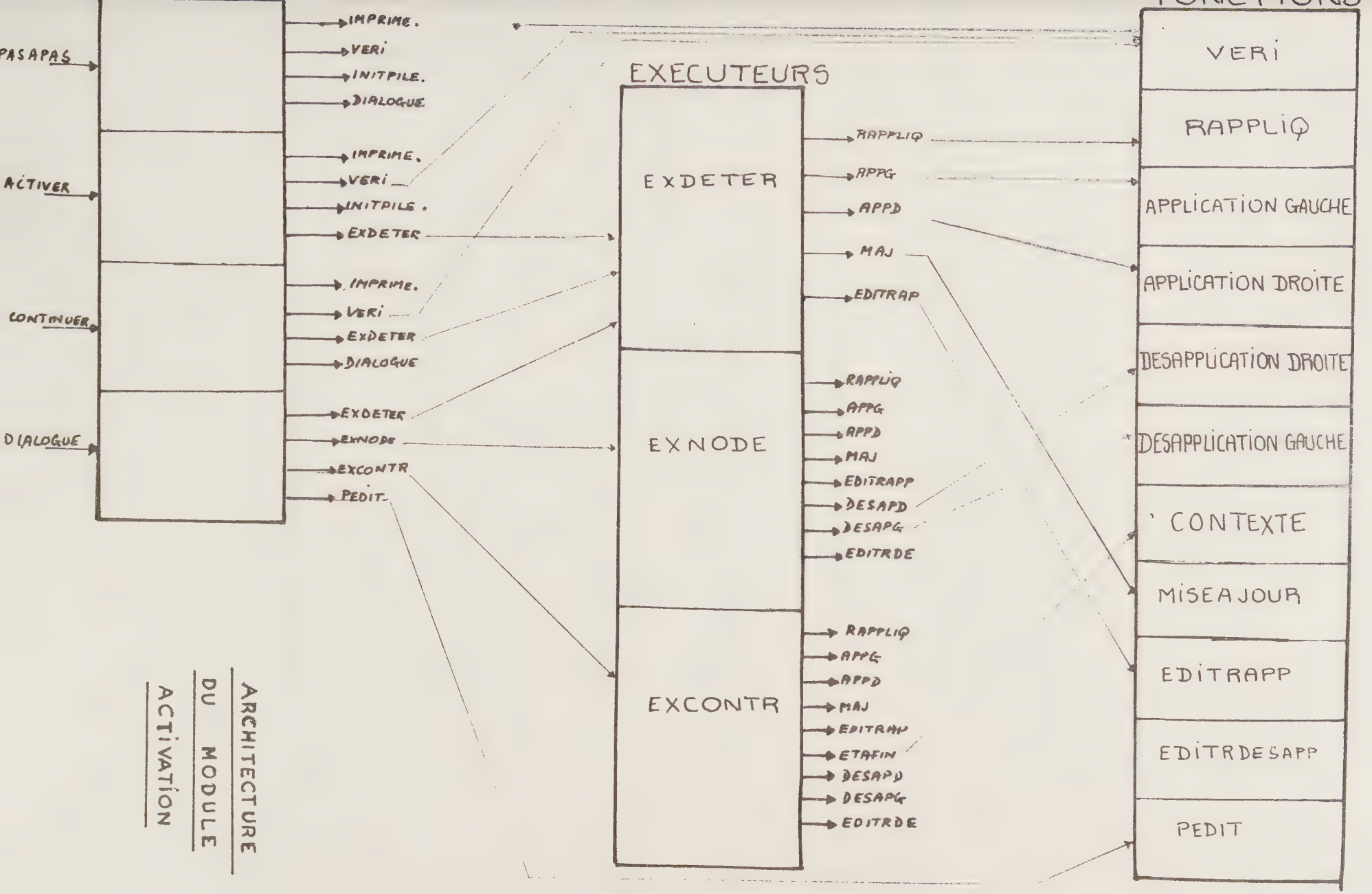
. demande à l'usager de choisir l'exécuteur en émettant :

DIALOGUE

. termine

REPONSE

REP = Code réponse de DIALOGUE



ARCHITECTURE DU MODULE D'ACTIVATION

REP = ACTIVER

Pas d'arguments

OBJET : traite la commande :

ACTIVER

par laquelle l'usager demande l'activation déterministe, sans éditions ni controle, d'un automate :

. vérifie si l'exécution est possible :

. vérifie qu'il existe bien un automate courant :

si COAUTO = 0 alors : . émet IMPRIME (PAS D'AUTOMAT

. termine

. vérifie le contexte en émettant :

VERI si REP = -1 termine

. vide la pile d'exécution : en émettant :

INITPILE

. positionne la cellule CONTPAS à 0 pour entrée future en CONTIN

. positionne les constantes d'éditions EDIC et EDIR :

. EDIC : = 2

. EDIR : = 2

car éditions non voulues.

. active l'exécuteur choisi en émettant :

EXDETER (EDIR, EDIC)

. termine

REPONSE

REP = code réponse de EXDETER

REP = CONTINUER

pas d'arguments

OBJET : traite la commande :

CONTINUER

par laquelle l'usager demande la poursuite de la dernière activation réalisée :

. vérifie si la poursuite de la dernière exécution est possible :

. vérifie qu'il existe bien un automate courant :

si COAUTO = 0 alors : . émet IMPRIME (PAS D'AUTOMATE)

. termine

. vérifie le contexte en émettant :

VERI si REP = -1 termine.

. lance l'exécuteur :

. Cas CONTPAS = 0 (entre précédente par ACTIVER)

. positionne les constantes d'éditions :

EDIC = 2

EDIR = 2 : éditions non voulues.

. active l'exécuteur déterministe en émettant :

EXDETER (EDIC, EDIR)

. Cas CONTPAS = 2 (entrée précédente par PASAPAS)

le choix de l'exécuteur doit être laissé à l'usager

on émet : DIALOGUE

. termine.

REPONSE : REP = code réponse de DIALOGUE ou de EXDETER selon le cas.

REP = DIALOGUE

Pas d'arguments

OBJET : choix de l'exécuteur par l'usager :

1) demande si le mode CONTROLE est voulu en émettant :
OUINON (CONTROLE,REPONSE)
. si REPONSE = NON : demande si le mode DETERMINISTE est voulu en émettant :
OUINON (EXECUTION DETERMINISTE, REPONSE)
. si REPONSE = OUI :
. demande les éditions voulues : émet :
PEDIT (EDIC, EDIR)
. active l'exécuteur déterministe : émet :
EXDETER (EDIC, EDIR)
. si REPONSE = NON :
. demande les éditions voulues : émet :
PEDIT (EDIC, EDIR)
. active l'exécuteur non déterministe: éme
EXNODE (EDIC, EDIR)
. si REPONSE = OUI :
. demande les éditions voulues : émet :
PEDIT (EDIC, EDIR)
. active l'exécuteur controlé : émet :
EXCONTR (EDIC, EDIR)
si REP = + 1 alors retourne en 1)
. termine

REPONSE
REP = code réponse des exécuteurs.

sous-module EXECUTEURS

OBJET : exécution d'un automate selon l'un des trois modes existant.

PRIMITIVES :

REP = EXDETER (EDIC, EDIR)

ARGUMENTS D'ACTIVATION

EDIC = 0 si édition du contexte désiré, 2 sinon
EDIR = 0 si édition de la règle désirée, 2 sinon

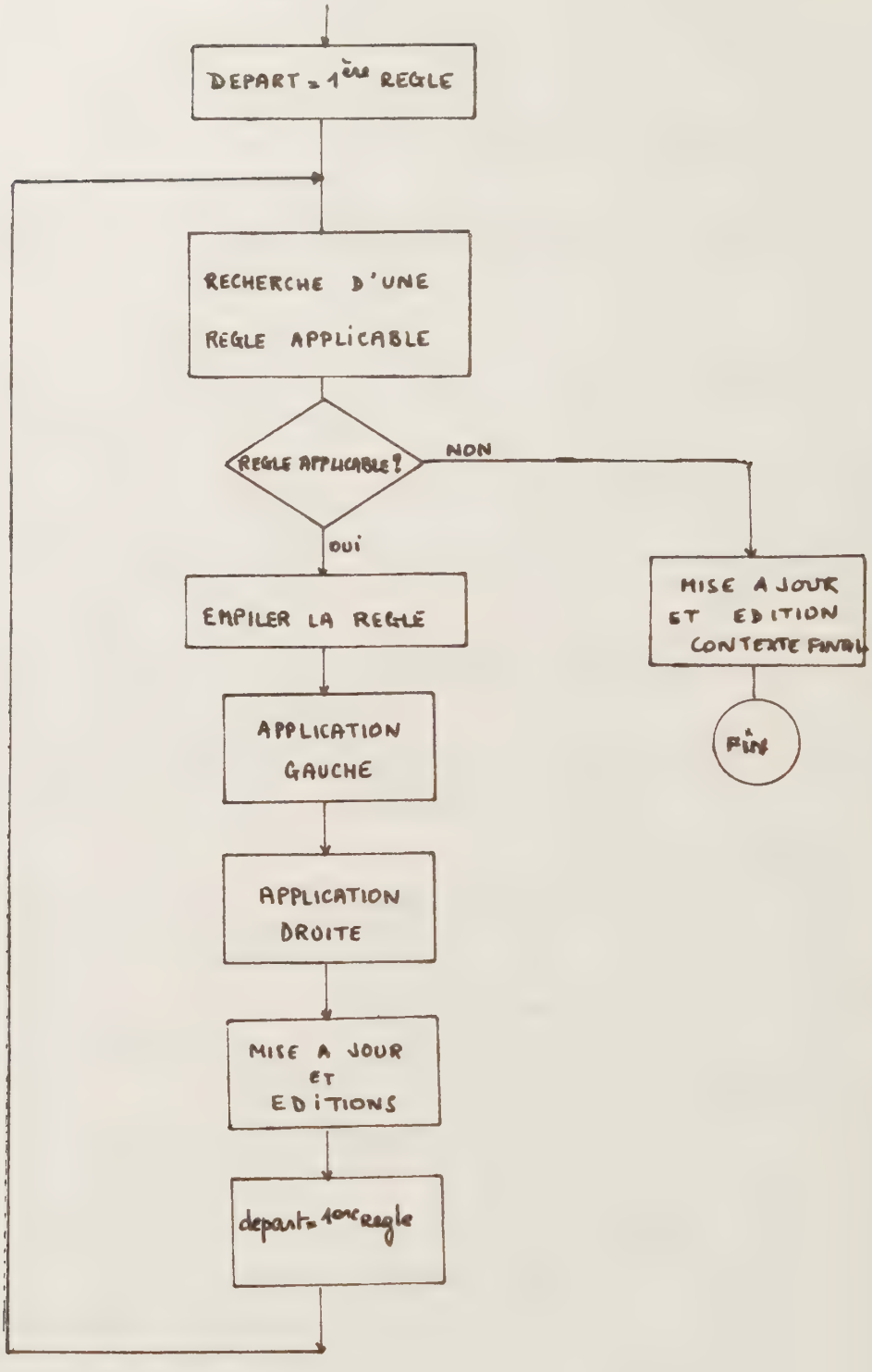
OBJET : exécute un automate selon le mode déterministe :
. recherche parmi les règles archivées la 1ere qui soit applicable
. modifie en fonction de cette règle la valeur des entités composant le contexte.
. cherche à appliquer une nouvelle règle.

REPONSE

REP = 0 si fin normale
REP = -2 si erreur durant l'exécution d'une fonction

STRUCTURE

Cf. page suivante.


DEPART = 1ère REGLE
RECHERCHE D'UNE REGLE APPLICABLE
REGLE APPLICABLE?
NON
OUI
EMPILER LA REGLE
APPLICATION GAUCHE
APPLICATION DROITE
MISE A JOUR ET EDITIONS
depart = 1ere regle
MISE A JOUR ET EDITION CONTEXTE FINAL
FIN

REP = EXNODE (EDIC, EDIR)

ARGUMENTS D'ACTIVATION :

EDIC = 0 si édition du contexte désiré, 2 sinon
EDIR = 0 si édition du contexte désiré, 2 sinon

OBJET : exécute un automate selon le mode non déterministe.

1). applique des règles jusqu'à ce qu'il n'y ait plus de règle applicable. Il y a alors deux cas à envisager : le contexte auquel on est arrivé peut être considéré comme satisfaisant ou non :
- si l'automate courant possède une entrée ou une pile ou les deux, un contexte satisfaisant est un contexte où l'entrée est finie ou bien ou la pile est vide
- si l'automate ne possède ni entrée ni pile il n'y a pas de contexte satisfaisant.

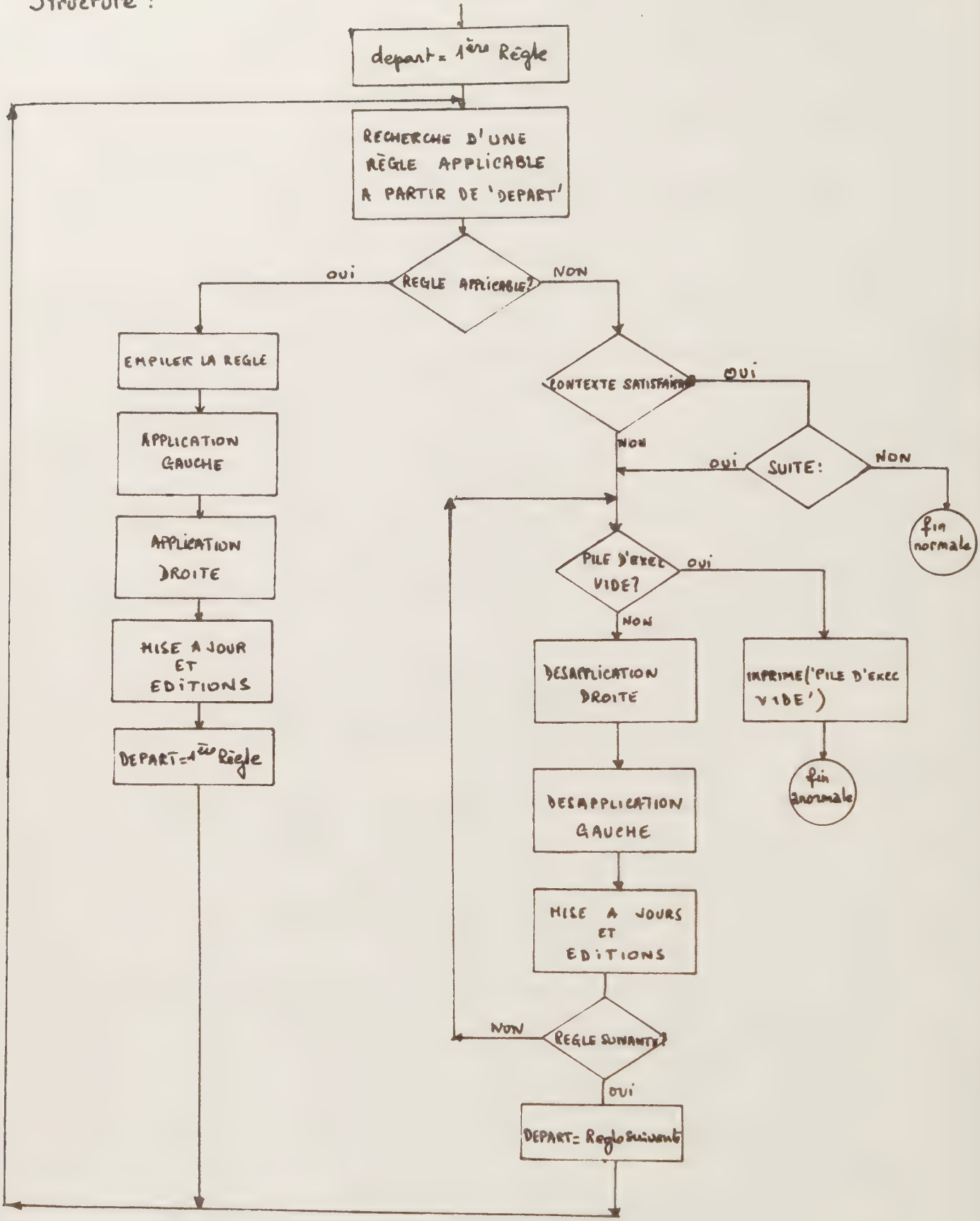
. si le contexte auquel on est arrivé n'est pas satisfaisant :
2). essaie de désappliquer la dernière règle appliquée :
. s'il n'y a plus de règles à désappliquer alors termine (REP = -1)
. sinon désapplique une règle et restitue le contexte
. cherche dans ce contexte une règle applicable à la suite de celle qui vient d'être désappliquée
. si une telle règle existe retourne en 1)
. sinon retourne en 2)

. si le contexte est satisfaisant demande à l'usager s'il veut désappliquer en émettant :
OUINON (SUITE,REPONSE)
si REPONSE = OUI retourne en 2

termine (REP = 0)

REPONSE
REP = 0 si fin normale
REP = -1 si fin anormale
REP = -2 si erreur durant l'exécution d'une fonction.

Structure :


depart = 1ère Règle
RECHERCHE D'UNE RÈGLE APPLICABLE A PARTIR DE 'DEPART'
REGLE APPLICABLE?
oui
NON
EMPILER LA REGLE
APPLICATION GAUCHE
APPLICATION DROITE
MISE A JOUR ET EDITIONS
DEPART = 1ère Règle
CONTEXTE SATISFAIT?
oui
NON
SUITE:
oui
NON
fin normale
PILE D'EXEC VIDE?
oui
NON
DESAPPLICATION DROITE
DESAPPLICATION GAUCHE
MISE A JOURS ET EDITIONS
REGLE SUIVANTE?
NON
oui
DEPART = Regle suivante
IMPRIME('PILE D'EXEC VIDE')
fin normale

REP = EXCONTR (EDIC, EDIR)

ARGUMENTS D'ACTIVATION :

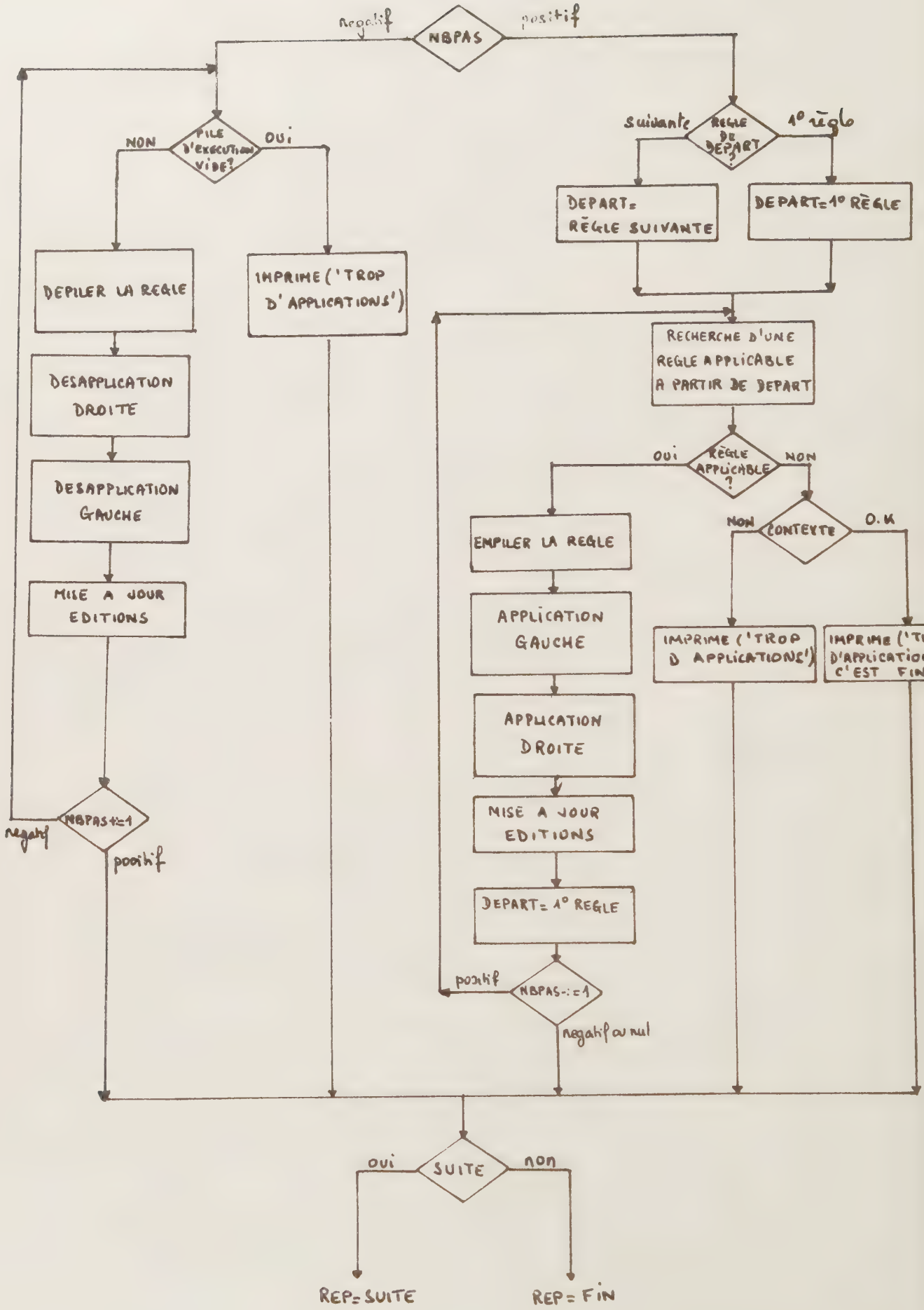
EDIC = 0 si édition du contexte désiré, 2 sinon
EDIR = 0 si édition de la règle désirée, 2 sinon

OBJET : exécute un automate selon le mode controlé :
. demande le nombre de pas : NBPAS
. désapplique ou applique le nombre de règles demandées par l'usager :
. si NBPAS < 0 alors désapplique :
. si impossible de désappliquer toutes les règles émet :
IMPRIME (TROP DE SESAPPLICATIONS)
. si NBPAS > 0 alors :
. demande la règle de départ qui peut être :
. soit la 1ere règle dans le cas d'un début d'exécution
. soit la règle suivante dans le cas d'une application après désapplication - parcours de l'ensemble des règles.
. applique :
. si impossible d'appliquer toutes les règles et si contexte satisfaisant émet :
IMPRIME (TROP D'APPLICATIONS)
IMPRIME (C'EST FINI)
. si le contexte n'est pas satisfaisant :
émet : IMPRIME (TROP D'APPLICATIONS)

a) . demande si SUITE désirée en émettant :
OUINON (SUITE, REPONSE) si REP = -1 retourne en a)
si REPONSE = OUI termine REP = 1
si REPONSE = NON termine REP = 0

REPONSE
REP = 0 si fin normale
REP = + 1 si suite désirée
REP = - 2 si erreur durant l'exécution d'une fonction.

Structure:

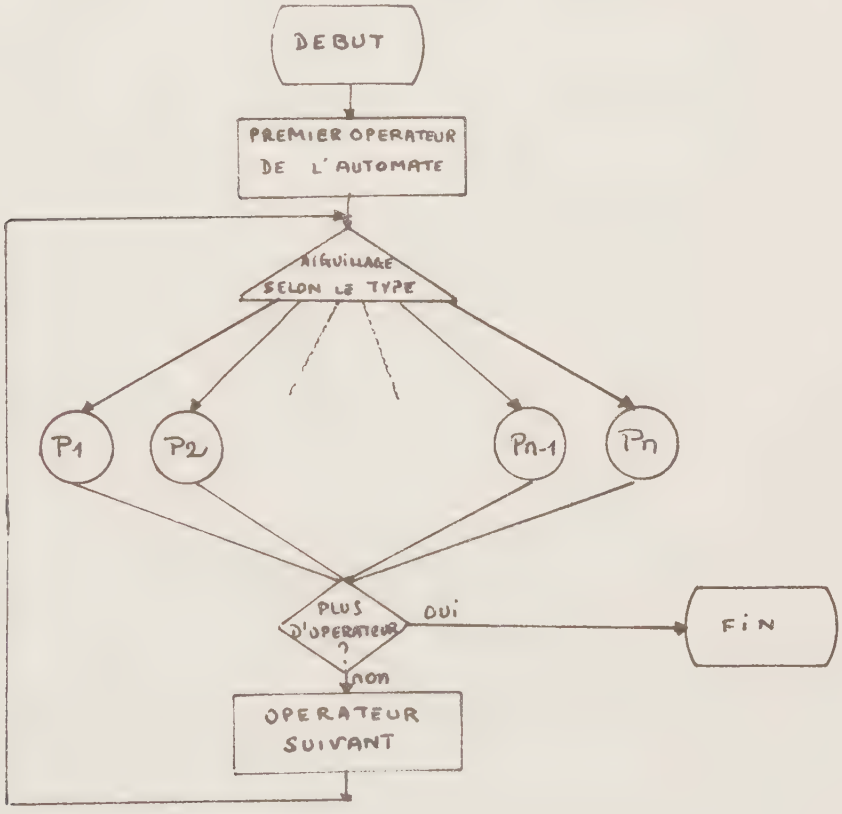

NBPAS
negatif
positif
PILE D'EXECUTION VIDE?
NON
OUI
DEPILER LA REGLE
DESAPPLICATION DROITE
DESAPPLICATION GAUCHE
MISE A JOUR EDITIONS
NBPAS+=1
negatif
positif
IMPRIME ('TROP D' APPLICATIONS')
REGLE DE DEPART?
suivante
1° règle
DEPART= REGLE SUIVANTE
DEPART=1° REGLE
RECHERCHE D'UNE REGLE APPLICABLE A PARTIR DE DEPART
REGLE APPLICABLE?
OUI
NON
EMPILER LA REGLE
APPLICATION GAUCHE
APPLICATION DROITE
MISE A JOUR EDITIONS
DEPART= 1° REGLE
positif
NBPAS-=1
negatif ou nul
CONTEXTE
NON
O.K
IMPRIME ('TROP D APPLICATIONS')
IMPRIME ('TRO D'APPLICATIONS C'EST FINI
SUITE
oui
non
REP= SUITE
REP= FIN

sous-module FONCTIONS D'EXECUTION

OBJET : regroupe les différentes fonctions qui apparaissent lors de l'exécution d'un automate.

PRIMITIVES : cf. pages suivantes.

STRUCTURES DES FONCTIONS : les différentes fonctions d'exécution, - sauf celle d'édition - et le test d'applicabilité d'une règle ont la même structure particulièrement simple :



où les Pi désignent les procédures, éventuellement vides, associées aux opérateurs pour la fonction considérée.

REP = VERI

Pas d'arguments

OBJET : initialise avant exécution : vérifie que le contexte courant est suffisant pour l'exécution de l'automate courant.

PROCEDURES : les procédures associées aux opérateurs pour cette fonction sont les suivantes :

- ETAT :
  - . si COETAT = 0 alors : émet : IMPRIME (PAS D'ETAT)
    sinon : émet : INITETAT
  - . termine
- ENTREE-AVANCE ENTREE :
  - . si COENTR = 0 alors : émet : IMPRIME (PAS D'ENTREE)
  - . termine
- PILE :
  - . si COPILE = 0 alors : émet : IMPRIME (PAS DE PILE)
  - . termine
- BANDE-AVANCE BANDE :
  - . si COBAND = 0 alors : émet : IMPRIME (PAS DE BANDE)
  - . termine
- SORTIE-AVANCE SORTIE :
  - . si COSORT = 0 alors : émet : IMPRIME (PAS DE SORTIE)
    sinon : émet : INITSORT
  - . termine

REPONSE

REP = 0 contexte suffisant

REP = -2 contexte insuffisant.

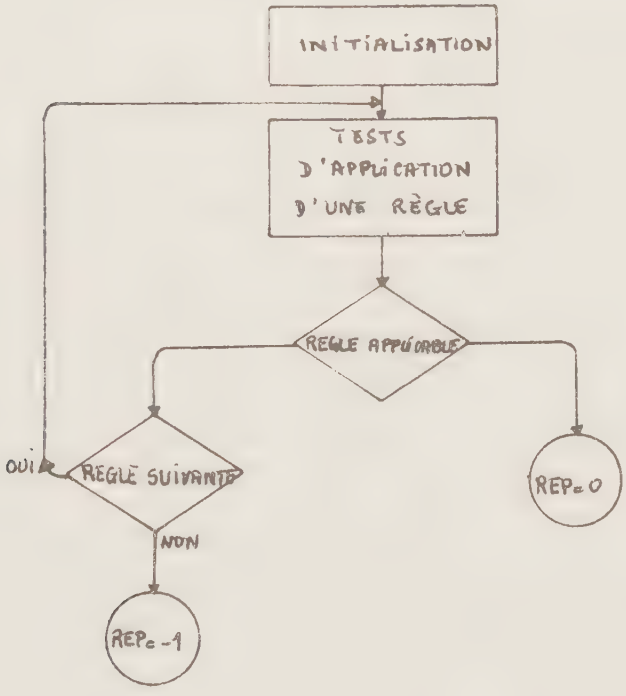
REP = RAPPLIQ (DEPART)

ARGUMENT D'ACTIVATION :

DEPART : référence de la règle à partir de laquelle on commence la recherche.

OBJET : recherche dans les règles archivées selon l'ordre lexical des champs la 1ère règle applicable qui suit la règle repérée par DEPART. Une règle est applicable si chacun des champs de son membre gauche est applicable.

STRUCTURE : la fonction de recherche d'une règle applicable se présente sous la forme suivante :



. INITIALISATION :

. initialise le tableau des caractères de la règle appartenant à 1 classe :

TABCLAS = 0

PTCLAS pointe sur TABCLAS [1]

. positionne 1 pointeur dans la pile émet :

STSAUV

. TEST D'APPLICABILITÉ: les procédures associées aux opérateurs se limitent à l'émission de la primitive correspondante :

- ETAT : ETCHA
- ENTREE-AVANCE ENTREE : ENCHA
- PILE : PICHA
- BANDE-AVANCE BANDE : BACHA

. REGLE APPLICABLE ? :

. si REP = - 2 termine (REP = - 2)

. si REP = 0 règle applicable empilée à partir de SAUV

. si REP = - 1 pas de règle applicable

. REGLE SUIVANTE :

. restauration de la table TABCLAS :

. TABCLAS : = 0

. PTCLAS pointe sur TABCLAS [1]

. back tracking : on remonte dans à la recherche d'un champ précédent applicable amenant à une nouvelle règle

. saut au test d'applicabilité.

REPONSE

REP = 0 il existe une règle applicable

REP = - 1 il n'existe pas de règle applicable

REP = - 2 erreur.

REP = APPLICATION GAUCHE

Pas d'arguments

PROCEDURES : les procédures associées aux opérateurs se limitent à l'émission de la primitive correspondante :

- ENTREE : EXGENTR
- PILE : EXGPILE

REP = APPLICATION DROITE

Pas d'arguments

PROCEDURES : les procédures associées aux opérations se limitent à l'émission de la primitive correspondante :

- ETAT = EXDET
- AVANCE ENTREE = EXDAVE
- PILE = EXDPILE
- BANDE = EXDBANDE
- AVANCE BANDE = EXDAVBA
- SORTIE = EXDSORTIE
- AVANCE SORTIE = EXDAVSO

REP = DESAP DROITE (DEPART)

ARGUMENT RESULTAT :

DEPART repére la règle suivante si elle existe pour RAPPLIQ dans le cas non déterministe.

PROCEDURES : les procédures associées aux opérateurs se limitent à l'émission de la primitive correspondante :

- ETAT : DESDET
- AVANCE ENTREE : DESDAE
- PILE : DESDPI
- BANDE : DESDAA
- AVANCE BANDE : DESDAB
- SORTIE : DESDSO
- AVANCE SORTIE : DESDAS

REP = DESAPGAUCHE (DEPART)

ARGUMENT RESULTAT :

DEPART : repère la règle suivante, si elle existe, pour RAPPLIQ dans le cas non déterministe.

PROCEDURES : les procédures associées aux opérateurs se limitent à l'émission de la primitive correspondante :

- ETAT : DESGET
- ENTREE : DESGEN
- PILE : DESGPI
- BANDE : DESGBA

REP = MISEAJOUR

Pas d'arguments.

PROCEDURES : les procédures associées aux opérateurs se limitent à l'émission de la primitive correspondante :

- ETAT : MAJET
- ENTREE : MAJEN
- PILE : MAJPI
- BANDE : MAJBA
- SORTIE : MAJSO.

REP = CONTEXTE (REPONSE)

ARGUMENT RESULTAT : REPONSE = OUI si contexte satisfaisant
REPONSE = NON si non

PROCEDURES : les procédures associées aux opérateurs sont les suivantes :

- ENTREE : VIDENT
- PILE : VIDPIL

REP = PEDIT (EDIC, EDIR)

ARGUMENTS RESULTAT :

. EDIC = 0 si édition du contexte voulue, 2 sinon
. EDIR = 0 si édition de la règle voulue, 2 sinon

OBJET : détermine les constantes d'édition :

a) . émet : OUINON (EDITION CONTEXTE, REPONSE)
. si REP = - 1 retourne en a)
. si REPONSE = OUI alors : EDIC = 2
sinon : EDIC = 0

b) . émet : OUINON (EDITION REGLE, REPONSE)
. si REP = - 1 retourne en a)
. si REPONSE = OUI alors : EDIR = 2
sinon : EDIR = 0

. termine

REPONSE :

REP = 0 fin normale.

REP = EDITRAPP

Pas d'arguments

OBJET : édition de la dernière règle appliquée :

. émet IMPRIME (REGLE APPLIQUEE :)
. édite la règle : émet :
PEDIR (SAUV, COAUTO)
. termine

REP = EDITDESAP

Pas d'arguments

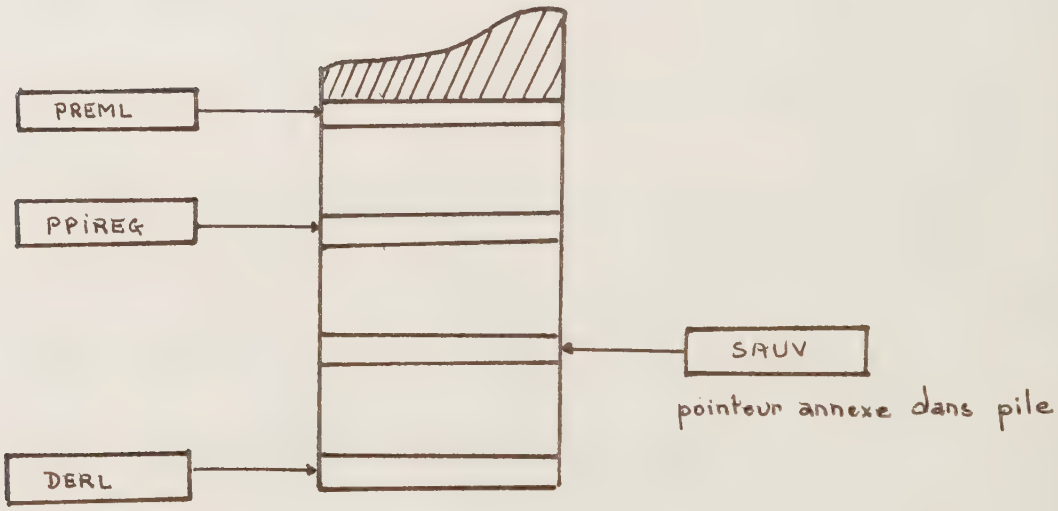
OBJET : édition de la dernière règle désappliquée :

. émet IMPRIME (REGLE DESAPPLIQUEE :)
. édite la règle : émet
PEDIR (SAUV, COAUTO)
. désalloue la règle désappliquée : émet :
POSSAUV
. termine.

MODULE PILEXEC

OBJET : gestion de la pile d'exécution.

STRUCTURE MANIPULEE : la pile PILEXEC est une pile de base DERL, de sommet repéré par PPIREG, de hauteur maximum DERL - PREML, dans laquelle sont placés les références des champs des règles appliquées lors d'une exécution :



REP = INITPILE

OBJET : restaure la pile d'exécution :

PPIREG = DERL

REP = EMPILCH (PTARB)

OBJET : mise en pile d'un champ repéré par un pointeur (PTARB)

si PPIREG PREML alors : PILEXEC (PPIREG) : = PTARB .termine REP = 0

sinon : émet IMPRIME (PLUS DE PLACE).termine REP =-2

REP = SETSAUV

OBJET : sauvegarde la valeur du pointeur PPIREG de la pile d'exécution

SAUV = PPIREG

REP = POSSAUV

OBJET : positionne le pointeur courant à l'endroit repéré par SAUV

PPIREG = SAUV

REP = PILVIDE (REPONSE)

OBJET : teste si pile vide :

. si PPIREG = DERL alors : REPONSE = OUI

sinon : REPONSE = NON

. fin

REP = PPIREG (POINTEUR, REPONSE)

OBJET : teste si le pointeur passé en argument est égal au pointeur courant PPIREG :

. si PPIREG = POINTEUR alors : REPONSE = OUI

sinon : REPONSE = NON

. fin

V-26. LA MACHINE OPERATEUR

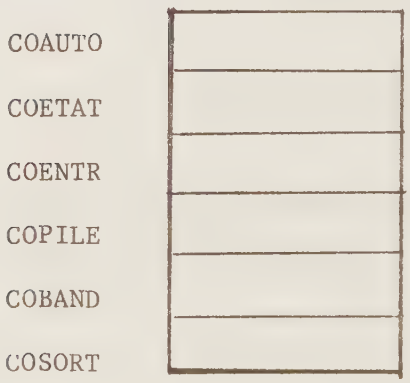
OBJET : gère les opérateurs existant dans SIMAC.

MODULES :

MODULE : TABLE DES OPERATEURS COURANTS

OBJET : gère la table des opérateurs courants.

STRUCTURE MANIPULEE : la table TOPCO : elle contient les références des opérateurs courants : chaque cellule pointe sur l'identificateur de l'opérateur courant :



PRIMITIVES : cf. pages suivantes.

REP = DETCO

Pas d'argument

OBJET : detruit l'opérateur qui est l'item courant de la liste des identificateurs :

. cherche l'adresse de l'item courant dans la liste des identificateurs en émettant :

COURANT (REFIDEN, REFSTRUC) si REP = - 1 termine

. recherche dans la table des opérateurs courants en émettant :

TABLE (REFIDENT, TOPCO, REFITEM)

. si REFITEM = - 1 alors termine

. détruit l'opérateur courant :

TOPCO [REFITEM] : = 0

. termine

REPONSE :

REP = 0 fin après destruction

REP = - 1 pas de destruction car pas d'item

REP = COUROPE (TYP, REFITEM)

ARGUMENTS D'ACTIVATION

TYP = N° de type de l'opérateur

REFITEM = référence de l'item courant de la liste des identificateurs

OBJET : prend pour opérateur courant l'item courant de la liste des identificateurs :

. vérifie que l'item courant est bien du type spécifié :

si NOTYP (REFITEM) ≠ TYP alors :

. émet : IMPRIME (INCOMPATIBLE)

. termine

. rend courant l'opérateur :

TOPCO [TYP] = REFITEM

. termine

REPONSE :

REP = 0 si fin normale

REP = - 1 si erreur

REP = EDITCONTEX

Pas d'arguments

OBJET : édite le contexte courant :

boucle sur table des opérateurs courants :

. initialisation : COMPT : = MAXOPERAT

1) . boucle :

. teste de fin de boucle :

. COMPT - : 1

. si COMPT < 0 alors termine.

. REF : = TOPCO [COMPT]

. si REF ≠ 0 alors : . place le nom et le type dans le tampon : EMPLIAMP (NOM (REF), LGNOTYP)

. édite le tampon : EDITAMP.

. retourne en 1)

REPONSE :

REP = 0 fin normale

MODULE TRAITEMENT DES OPERATEURS.

OBJET : Il s'agit des procédures associées aux fonctions d'exécution pour chaque opérateur.

STRUCTURE DU MODULE : elle apparaît dans le tableau suivant :

FONCTION D'EXECUTION	OPERATEURS				
	ETAT	ENTREE	PILE	BANDE	SORTIE
VERI	INITETAT	Ø	Ø	Ø	INITSORT
RAPPLIQ	ETCHA	ENCHA	PICHA	BACHA	Ø
APP GAUCHE	Ø	EXGENTR	EXGPILE	Ø	Ø
APP DROITE	EXDET	EXDAVE	EXDPILE	EXBANDE	EXDSORTIE
				EXAVBA	EXDAVSO
DES DROITE	DESDET	DESDAE	DESPI	DESDBA	DESDAS
				DESDAB	DESDSO
DES GAUCHE	DESGET	DESGEN	DESGPI	DESGBA	Ø
CONTEXTE	Ø	VIDENT	VIDPIL	Ø	Ø
MISEAJOUR	MAJET	MAJEN	MAJPI	MAJBA	MAJSO

REP = INITETAT

Pas d'arguments

OBJET : initialise l'état COURANT avant exécution :

. calcule l'adresse du descriptif de l'état :

REF : = REF (COETAT)

. modifie l'état courant : place le pointeur courant en tête de variable :

. COURANT (REF) : = DEBUT (REF)

. APRES (REF) : = TOTAL (REF)

. AVANT (REF) : = 0

. termine

REPONSE :

REP = 0 fin normale

REP = INITSORT

Pas d'arguments

OBJET : initialise la sortie COURANT avant exécution :

. calcule l'adresse du descriptif de la sortie :

REF : = REF (COSORT)

. modifie la sortie courante : place le pointeur fin sur le caractère repéré par le pointeur courant :

. FIN (REF) : = COURANT (REF)

. TOTAL (REF) : = AVANT (REF)

. APRES (REF) : = 0

REPONSE :

REP = 0 fin normale

REP = ETCHA (DEPART, COURANT, APRES, NVET)

ARGUMENTS D'ACTIVATION :

DEPART : tête de la liste des suivants des champs d'état.
COURANT : pointeur courant sur la chaîne.
APRES : longeur maximum de la chaîne
RESULTAT :
DEPART = NVET = référence du champ trouvé.

OBJET : recherche dans la liste si le contenu d'un des champs est identique à l'état courant :

. boucle sur les champs
a).teste si fin de liste :
. DEPART : = SUIV (DEPART)
. si DEPART = 0 alors termine (REP = -1)

. teste si comparaison possible :
si EFFECT (DEPART) ≠ APRES alors : retourne en a)

. compare les champs : si NOMBCAR (DEPART) ≠ EFFECT (DEPART)
alors : émet COMPCLA (DEPART,COURANT, LGZ)
si REP = -1 retourne en a)
sinon : émet COMPCHA (DEPART, COURANT, LGZ)
si REP = -2 termine (REP = -2)
si REP = -1 retourne en a)

. NVET : = DEPART
. met en pile le champ en émettant : EMPILCH (DEPART)
si REP = -2 termine (REP = -2)
. termine (REP = 0)

REPONSE :
REP = 0 champ trouvé
REP = - 2 si erreur
REP = - 1 fin de liste.

REP = ENCHA (DEPART, COURANT, APRES, NVEN, NBEN)

ARGUMENTS D'ACTIVATION :

DEPART = tête de la liste des suivants des champs entrée
COURANT = pointeur courant sur la chaîne
APRES = longueur maximum de la chaîne

RESULTAT :

DEPART = référence du champ trouvé
NVEN = nouvelle entrée ; NBEN = nombre de caractères reconnus

OBJET : recherche dans la liste si le contenu de l'un des champs figure sur l'entrée courante immédiatement après la tête :

. boucle sur les champs :
a).teste si fin de liste :
. DEPART : = SUIV (DEPART)
. si DEPART = 0 alors termine (REP = - 1)

. teste si comparaison possible :
si EFFECT (DEPART) = APRES alors retourne en a)
. compare les champs : si NOMBCAR (DEPART) ≠ EFFECT (DEPART)
alors : émet COMPCLA (DEPART, COURANT, LG8)
si REP = - 1 retourne en a)
sinon : émet COMPCHA (DEPART, COURANT, LGZ)
si REP = - 2 termine (REP = - 2)
si REP = - 1 retourne en a)

. calcule le nouveau pointeur courant :
NVEN : = COURANT + LGZ ; NBEN : = LGZ
. met en pile le champ en émettant : EMPILCH (DEPART)
si REP = - 2 termine (REP = - 2)
. termine.

REPONSE : REP = 0 champ trouvé
REP = - 2 si erreur
REP = - 1 fin de liste.

REP = PICHA (DEPART, COURANT, AVANT, NVPI, NBPI)

ARGUMENTS D'ACTIVATION :

DEPART = tête de la liste des suivants des champs pile.
COURANT = pointeur courant sur la chaîne
AVANT = longueur maximum de la chaîne

RESULTAT :

DEPART = réference du champ trouvé
NVPI = nouvelle pile
NBBA = nombre de caractères reconnus.

OBJET : recherche dans la liste si le contenu de l'un des champs figure sur l'entrée courante immédiatement après la tête :

. boucle sur les champs :

a).teste si fin de liste :
. DEPART : = SUIV (DEPART)
. si DEPART = 0 alors termine (REP = - 1)

. teste si comparaison possible :
si EFFECT (DEPART) > APRES alors retourne en a)

. compare les champs : si NOMBCAR (DEPART) ≠ EFFECT (DEPART)
alors : émet COMPCLA (DEPART, COURANT, LG8)
si REP = - 1 retourne en a)
sinon : émet COMPCHA (DEPART, COURANT, LGZ)
si REP = - 2 termine (REP = - 2)
si REP = - 1 retourne en a)

. calcule le nouveau pointeur courant :
NVEN : = COURANT + LGZ ; NBEN : = LGZ

. met en pile le champ en émettant : EMPILCH (DEPART)
si REP = - 2 termine (REP = - 2)

. termine (REP = 0)

REPONSE :
REP = 0 champ trouvé
REP = - 2 si erreur
REP = - 1 fin de liste.

REP = BACHA (DEPART, COURANT, APRES, NVBA, NBBA)

ARGUMENTS D'ACTIVATION :

DEPART = tête de la liste des champs bande
COURANT = pointeur sur la chaine
APRES = longueur maximum de la chaine

RESULTAT

DEPART = référence du champ trouvé
NVBA = nouvelle bande
NBBA = nombre de caractères reconnus.

OBJET : recherche dans la liste si le contenu de l'un des champs figure sur la bande courante immédiatement après la tête :

. boucle sur les champs :
a) . teste si fin de liste :
. DEPART : = SUIV (DEPART)
. si DEPART = 0 alors termine (REP = - 1)
. teste si comparaison possible :
si EFFECT (DEPART) > APRES alors retourne en a)
. compare les champs :
si NOMBCAR (DEPART) ≠ EFFECT (DEPART)
alors : émet COMPCLA (DEPART, COURANT, LGZ)
. si REP = - 1 retourne en a)
sinon : émet COMPCHA (DEPART, COURANT, LGZ)
si REP = - 2 termine (REP = - 2)
si REP = - 1 retourne en a)
. calcule le nouveau pointeur courant :
NVBA : = COURANT + LGZ ; NBBA : = LGZ
. met en pile le champ en émettant : EMPILCH (DEPART)
si REP = - 2 termine (REP = - 2)
. termine (REP = 0)

REPONSE :
REP = 0 champ trouvé
REP = - 2 si erreur
REP = - 1 si fin de liste.

REP = COMPCHA (DEPART, COURANT, LGZ)

ARGUMENTS D'ACTIVATION :

DEPART = adresse du champ
COURANT = pointeur dans chaîne

RESULTAT

DEPART = inchangé
LGZ = nombre de caractères reconnus.

OBJET : compare un champ avec une chaîne : boucle sur les caractères de la zone de champ :

. initialise avant boucle :
LGZ : = NOMBCAR (DEPART)
PTCHAINE : = COURANT
PTZONE : = ZONE (DEPART)

a) . boucle :
. décremente : LGZ : LGZ - 1
. test si fin : si LGZ = 0 alors termine (REP = 0)
sinon :
. CAR : = caractère repéré par PTCHAINE
. si CAR = ↵ alors termine (REP : - 1)
sinon : si CAR ≠ ZONE (PTZONE)
alors : . PTCHAINE + = 1
. PTZONE + = 1
. retourne en a)
sinon : termine (REP = - 1)

REPONSE

REP = 0 le champ appartient à la chaîne
REP = - 1 le champ n'appartient pas à la chaîne

REP = COMPCLA (DEPART, COURANT, LGZ)

ARGUMENTS D'ACTIVATION :

DEPART = adresse du champ
COURANT = pointeur dans chaîne

RESULTAT

DEPART = inchangé
LGZ = nombre de caractères reconnus

OBJET : compare un champ contenant des classes de caractères avec une chaîne :

. initialise avant boucle :
. initialise le compteur de classe du champ : CHACLA : = 0
. NBTOT : = NOMBCAR (DEPART)
. LGZ : = EFFECT (DEPART)
. PTCHAINE : = COURANT
. PTZONE : = ZONE (DEPART)

. boucle sur le nombre total de caractères du champ
1) . Teste si fin de boucle :
. NBTOT - = 1
. si NBTOT ≤ 0 alors : termine (REP = 0)
sinon : . initialise boucle sur classe :
. TRCLAS = 2
. CAR : = caractère repéré par PTCHAINE
. si CAR = blanc alors :
2) . restaure TABCLAS :
. TABCLAS - = CHACLA
. PTCLAS - = CHACLA
. termine. (REP = - 1)

. boucle sur zone :
3) . CARZONE : = caractère repéré par PTZONE
. Cas TRCLAS = 2 :
si CARZONE ≠ '
alors : si CAR = CARZONE alors retourne en 1)
sinon retourne en 2)

sinon : initialisation classe :
. TRCLAS : = 0 (classe en cours)
. PTNOM pointe sur ZNOM
. NBCAR = NBCNOM
. NBTOT = NBTOT - 1
. retourne en 3)

. Cas TRCLAS = 0 :
si CARZONE ≠ '
alors : . NBCAR - = NBCAR
. si NBCAR 0 alors :
. ZNOM PTNOM : = CARZONE
. PTNOM + = 1
. NBTOT - = 1
. retourne en 3)
sinon : on complète par des blancs :
. CARZONE : = blanc
4) . BNCAR - = 1
. si NBCAR ⩾ 0 alors :
. ZNOM [PTNOM] : = CARZONE
. PTNOM + = 1
. retourne en 4)

. recherche la classe dont le nom est dans ZNOM
en émettant :
CHCLAS (ZNOM)
si REP = - 1 : . émet IMPRIME (CLASSE < NOM >
N'EXISTE PAS)
. termine (REP = - 2)

. Cherche si le caractère courant de la chaîne
appartient à la classe courante :
. émet COURANT (REFITEM, REFSTRUC)
. émet CHCAR (REFSTRUC, CAR, NOMOT, RESTE, CARES)
si CARES = 0 alors retourne en 2)
sinon : place le caractère dans TABCLAS :
. TABCLAS [PTCLAS] : = CAR
. TABCLAS + = 1
. PTCLAS + = 1
. CHACLA + = 1
. retourne en 1)

REPONSE

REP = 0 si le champ $\in$ chaine

REP = - 1 si le champ $\notin$ chaine

REP = - 2 si la classe reconnue n'existe pas.

REP : EXGENTR (NVEN, NBBEN)

ARGUMENTS D'ACTIVATION :

NVEN = nouvelle entrée

NBBEN = nombre de caractères reconnus

OBJET : déplacement de la tête

. COURANT (COENTR) : = NVEN

. APRES (COENTR) - = NBEN

. termine

REPONSE

REP = 0 fin normale

REP = EXGPILE (NVPI, NBBI)

ARGUMENTS D'ACTIVATION

NVPI = nouvelle pile

NBBI = nombre de caractères reconnus

OBJET : déplacement de la tête :

. COURANT (COPILE) : = NVPI

. APRES (COENTR) + = NBPI

. termine.

REPONSE

REP = 0 fin normale.

REP = EXDET (NVET)

ARGUMENT D'ACTIVATION :

NVET = Référence du champ trouvé

OBJET : change la variable d'état :

. si NOMBCAR (NVET) = 0 alors termine (REP = 0)

. si NOMBCAR (NVET) NBCMAX (COETAT) alors :

. émet IMPRIME (DOC ETAT)

1) . désalloue : émet POSSAUV

. termine (REP = - 2)

sinon : . TOTAL (COETAT) : = NOMBCAR (NVET)

. traite le variable :

. PTCHAINE : = DEBUT (COETAT)

. PTZONE : = NVET

. émet : REBOP (PTCHAINE, PTZONE, CAREN)

si REP = - 2 : . émet IMPRIME (ERREURCLASSE)

. retourne en 1)

. termine.

REPONSE

REP = 0 fin normale

REP = - 2 si erreur.

REP = EXDAVE (REFCHAMP)

ARGUMENT D'ACTIVATION :

REFCHAMP = référence du champ.

OBJET : déplace la tête de l'entrée de la ligne précisée par le champ avance :

. émet EXDCHAV (REFCHAMP, COENTR, CAREN)

si REP = - 2 : . émet IMPRIME (DOC ENTREE)
. désalloue émet : PSSSAUV
. termine.

si REP = 0 : . COURANT (COENTR) + = CAREN
. APRES (COENTR) - = CAREN
. termine

REPONSE

REP = 0 fin normale

REP = - 2 si erreur.

REP = EXDPILE (REFCHAMP)

ARGUMENT D'ACTIVATION :

REFCHAMP = référence du champ.

OBJET : écrit le champ pile sur la file, avance la tête :

. si TOTAL (COPILE) + EFFECT (REFCHAMP) NBCMAX (COPILE)

alors : . émet IMPRIME (DDC PILE)

1) . désalloue : émet : POSSAUV

. termine (REP = - 2)

sinon : . TOTAL (COPILE) + = EFFECT (REFCHAMP)

. traite la pile :

. PTCHAINE : = FIN (COPILE)

. PTZONE : = REFCHAMP

. émet RECOP (PTCHAINE, PTZONE, CAREN)

si REP = - 2 : émet :

IMPRIME (ERREUR CLASSE)

retourne en 1)

si REP = 0 : FIN (COPILE) + = CAREN

. termine.

REPONSE

REP = 0 fin normale

REP = - 2 si erreur.

REP = EXDBANDE (REFCHAMP)

ARGUMENT D'ACTIVATION :

REFCHAMP = référence du champ.

OBJET : exécution droite d'une bande : écrit le champ et déplace la tête

. émet BANDE (REFCHAMP, PTCHAINE, CAREN)

si REP = -2 termine (REP = - 2)

. COURANT (COBAND) : = PTCHAINE

. APRES (COBAND) - = CAREN

. AVANT (COBAND) + = CAREN

. termine.

REPONSE

REP = 0 si fin normale

REP = - 2 si erreur.

REP = EXDAVBA (REFCHAMP)

ARGUMENT D'ACTIVATION :

REFCHAMP = référence du champ.

OBJET : exécution droite d'un champ avance bande : écrit le champ et déplace la tête :

. émet BANDE (REFCHAMP, PTCHAINE, CAREN)
si REP : - 2 termine (REP = - 2)
. COURANT (COBAND) : = PTCHAINE
. APRES (COBAND) - = CAREN
. AVANT (COBAND) + = CAREN
. émet EXDCHAV (FERCHAMP, COBAND, CAREN)
si REP = -2 alors : . émet IMPRIME (ODC BANDE)
. désalloue : émet POSSAUV
. termine (REP = - 2)
. COURANT (COBAND) + = CAREN
. APRES (COBAND) - = CAREN
. AVANT (COBAND) + = CAREN
. termine.

REPONSE

REP = 0 si fin normale
REP = - 2 si erreur.

REP = BANDE (REFCHAMP, PTCHAINE, CAREN)

ARGUMENT D'ACTIVATION :

REFCHAMP = référence du champ

RESULTAT :

PTCHAINE = nouveau pointeur courant

CAREN = nombre de caractères lus.

OBJET : écrit le champ bande :

. si AVANT (COBANDE) + EFFECT (REFCHAMP) NBCMAX (COBANDE)

alors : . émet IMPRIME (DOCBANDE)

1). désalloue : émet : POSSAUV

. termine (REP = - 2)

sinon : . traite la bande :

. PTCHAINE : = COURANT (COBAND)

. PTZONE : = REFCHAMP

. émet RECOP (PTCHAINE, PTZONE, CAREN)

si REP = - 2 : . émet IMPRIME (ERREUR CLASSE)

. retourne en 1)

. termine

REPONSE

REP ≠ 0 si fin normale

REP = - 2 si erreur.

REP = EXDSORTIE (REFCHAMP)

ARGUMENT D'ACTIVATION

REFCHAMP = référence du champ.

OBJET : exécution droite d'une sortie : écrit le champ et déplace la tête :

. émet SORTIE (REFCHAMP, PTCHAINE, CAREN)

si REP = - 2 termine (REP = - 2)

. déplace la tête :

. COURANT (COSORT) : = PTCHAINE

. AVANT (COSORT) + = CAREN

. APRES (COSORT) + = CAREN

. termine.

REPONSE

REP = 0 si fin normale

REP = − 2 si erreur.

REP = EXDAVSO (REFCHAMP)

ARGUMENT D'ACTIVATION :

REFCHAMP = référence du champ.

OBJET : exécution droite d'un champ avance sortie : écrit le champ et déplace la tête :

. émet SORTIE (REFCHAMP, PTCHAINE, CAREN)
si REP = – 2 termine (REP = - 2)

. déplace la tête :
. COURANT (COSORT) : = PTCHAINE
. AVANT (COSORT) + = CAREN
. APRES (COSORT) + = CAREN

. émet EXDCHAV (REFCHAMP, COSORT, CAREN)
si REP = - 2 alors : . émet IMPRIME (DDC SORTIE)
. désalloue : émet : POSSAUV
. termine (REP = - 2)

. déplace la tête :
. COURANT (COSORT) + = CAREN
. APRES (COSORT) - = CAREN
. AVANT (COSORT) + = CAREN

. termine.

REPONSE

REP = 0 si fin normale
REP = - 2 si erreur.

REP = SORTIE (REFCHAMP, PTCHAINE, CAREN)

ARGUMENT D'ACTIVATION :

REFCHAMP = référence du champ

RESULTAT :

PTCHAINE = nouveau pointeur courant

CAREN = nombre de caractères lus.

OBJET : écrit le champ sortie :

. si TOTAL (COSORT) + EFFECT (REFCHAMP) > NBCMAX (COBORT)

alors : . émet IMPRIME (DDC SORTIE)

1) . désalloue : émet : POSSAUV

. termine (REP = - 2)

sinon : . traite la sortie :

. PTCHAINE : = FIN (COSORT)

. PTZONE : = REFCHAMP

. émet RECOP (PTCHAINE, PZZONE, CAREN)

si REP = - 2 : . émet IMPRIME (ERREUR CLASSE)

. retourne en 1)

. termine.

REPONSE :

REP = 0 si fin normale

REP = - 2 si erreur.

REP = EXDCHAV (REFCHAMP, REFOPER, CAREN)

ARGUMENT D'ACTIVATION :

REFCHAMP = référence du champ avance

REFOPER = référence de l'opérateur courant.

OBJET : réalise l'exécution droite d'un champ avance :

. CAREN : = EFFECT (REFOPER)

. si CAREN + AVANT (REFOPER) 0 alors termine : (REP = - 2)

. si CAREN + AVANT (REFOPER) - NBCMAX (REFOPER) < 0 alors :

termine (REP = - 2)

sinon : termine (REP = 0)

REPONSE

REP = 0 si fin normale

REP = - 2 si erreur.

REP = RECOP (PTZONE, PTCHAINE, CAREN)

ARGUMENT D'ACTIVATION :

PTZONE = pointeur sur champ

PTCHAINE = pointeur sur chaine

RESULTAT :

CAREN = nombre de caractères ajoutés

PTCHAINE = nouveau pointeur sur chaine, = 0 si erreur

OBJET : reconnaît des classes dans un champ droit d'une règle, met en place les caractères correspondants et les ajoute à la chaîne :

. Initialisation :

. NBTOT : = NOMBCAR (PTZONE)

. EFFECT : = EFFECT (PTZONE)

. Reconnaît les classes : si NBTOT ≠ EFFECT alors :

. initialise avant boucle sur NBTOT :

. PTZONE pointe sur ZONE (PTZONE)

. si TABCLAS = 0 alors :

1) . PTCHAINE : = 0

. termine (REP = - 2)

. TRCLAS : = - 2

. initialise la zone reconstituée :

. PTZONE pointe sur ZONREC [1]

. ZONREC : = 0

. boucle sur NBTOT :

. teste si fin de liste :

. NBTOT - = 1

. si NBTOT ⩾ 0 alors : saute en 2)

. cas TRECLAS = 0 :

. si CAR ≠ ' alors : . ZONOMB + = 1

. ZONOMB [PTZOMB] : = CAR

sinon : . TRCLAS : = 2

. convertit le nombre en émettant :

NOMBRE (ZONOMB [1], ZONOMB)

si REP = - 1 alors retourne en 1)

si ZONOMB [1] ≤ 0 alors retourne en 1)

si TABCLA < ZONOMB [1] alors retourne en 1)

. met en place le caractère :

ZONREC [PTZONR] : = TABCLAS [ZONOMB [1]]

. incrémente compteur et pointeur :

. ZONREC + = 1

. PTZONR + = 1

. cas TRCLAS = 2

. si CAR ≠ ' alors : . ZONREC PTZONR : = CAR

. ZONREC + = 1

sinon . PTZOMB pointe sur ZONOMB + 1

. ZONOMB = 0

. TRCLAS = 0

. ajoute à la chaîne reperée par PTCHAINE : en émettant :

ADDCH (PTCHAINE, PTZONE, ZONREC, CAREN)

. termine.

REPONSE

REP = 0 si fin normale

REP = - 2 si erreur.

REP = DESDET (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT : DEPART pointe sur champ suivant.

OBJET : désapplication droite d'un champ état :

. émet : CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. termine (REP = 0)

REPONSE :

REP = 0 fin normale

REP = - 1 : DEPART pointe sur champ suivant.

REP = DESDAE (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART = pointe sur champ suivant.

OBJET : désapplication droite d'un champ avance entrée :

. INC : = NOMBCAR (REFCHAMP)

. Teste si désapplication possible :

si AVANT (COENTR) - INC < 0 alors termine (REP = - 2)

. Désapplique :

. COURANT (COENTR) - = INC

. APRES (COENTR) + = INC

. AVANT (COENTR) - = INC

. Emet : CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. Termine (REP = 0)

REPONSE

REP : 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur.

REP = DESDPI (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART = pointe sur champ suivant.

OBJET : désapplication droite d'un champ pile :

. Teste si désapplication possible :

. si NOMBCAR (REFCHAMP) ≠ EFFECT (REFCHAMP) alors termine

. si TOTAL (COPILE) - EFFECT (REFCHAMP) < 0 alors termine

. Désapplique :

. INC : = NOMBCAR (REFCHAMP)

. met à jour le descriptif :

. TOTAL (COPILE) - = INC

. AVANT (COPILE) - = INC

. COURANT (COPILE) - = INC

. FIN (COPILE) - = INC

. émet : CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. termine (REP = 0)

REPONSE

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur.

REP = DESDAB (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART : pointe sur champ suivant.

OBJET : désapplication droite d'un champ avance bande :

. INC : = NOMBCAR (REFCHAMP)

. Teste si désapplication possible :

si AVANT (COBAND) - INC < 0 alors termine (REP = - 2)

. Désapplique :

. COURANT (COBAND) - = INC

. APRES (COBAND) + = INC

. AVANT (COENTR) - = INC

. Emet CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. termine (REP = 0)

REPONSE :

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur

REP = DESDBA (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART = pointe sur champ suivant.

OBJET : désapplication droite d'un champ bande :

. Teste si désapplication possible :

. si NOMCAR (REFCHAMP) ≠ EFFECT (REFCHAMP) alors termine (REP = - 2)

. si AVANT (BANDE) - EFFECT (REFCHAMP) < 0 alors termine (REP = - 2)

. Désapplique :

. INC : = NOMBCAR (REFCHAMP)

. met à jour le descriptif :

. APRES (BANDE) + = INC

. AVANT (BANDE) - = INC

. COURANT (BANDE) - = INC

. Emet : CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. termine (REP = 0)

REPONSE :

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur

REP = DESDAS (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART pointe sur champ suivant.

OBJET : désapplication droite d'un champ avance sortie :

. INC : = NOMBCAR (EFFECT)

. Teste si désapplication possible :

si AVANT (COSORT) - INC < 0 alors termine (REP = - 2)

. Désapplique :

. COURANT (COSORT) - = INC

. APRES (COSORT) + = INC

. AVANT (COSORT) - = INC

. Emet : CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. Termine (REP = 0)

REPONSE :

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur.

REP = DESDSO (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART = pointe sur champ suivant.

OBJET : désapplication droite d'un champ sortie :

. Teste si désapplication possible :

. si NOMBCAR (REFCHAMP) ≠ EFFECT (REFCHAMP) alors termine (REP = - 2)

. si TOTAL (COSORT) - EFFECT (REFCHAMP) > 0 alors termine (REP = -2)

. Désapplique :

. INC : = NOMBCAR (REFCHAMP)

. met à jour le descriptif :

. TOTAL (COSORT) - = INC

. AVANT (COSORT) - = INC

. COURANT (COSORT) - = INC

. FIN (COSORT) - = INC

. Emet : CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. Termine (REP = 0)

REPONSE :

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur.

REP = DESGET (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART = pointe sur champ suivant.

OBJET : désapplication gauche d'un champ état :

. Teste si désapplication possible :

si NOMCAR (REFCHAMP) - EFFECT (REFCHAMP) ≠ 0 alors termine (REP = - 2)

. Désapplique :

. PTZONE : = REFCHAMP

. PTCHAINE : = DEBUT (COETAT)

. LGZ : = NOMBCAR (REFCHAMP)

. émet ADDCH (PTCHAINE, PTZONE, LGZ, CAREN)

. Met à jour le descriptif :

. FIN (COETAT) : = CAREN

. COURANT (COETAT) := CAREN

. AVANT (COETAT) : = LGZ

. TOTAL (COETAT) : = LGZ

. APRES (COETAT) : = 0

. Emet CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. Termine.

REPONSE :

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur.

REP = DESGEN (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART = pointe sur champ suivant.

OBJET : désapplication droite d'un champ entrée :

. Teste si désapplication possible :

. si NOMBCAR (REFCHAMP) - EFFECT (REFCHAMP) ≠ 0 alors termine

. si AVANT (COENTR) - EFFECT (CHAMP) < 0 alors termine (REP = -2)

. Met à jour le descriptif :

. INC : = NOMBCAR (REFCHAMP)

. APRES (COENTR) + = INC

. AVANT (COENTR) - = INC

. COURANT - = INC

. Emet CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. Termine.

REPONSE :

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur.

REP = DESGPI (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART = pointe sur champ suivant.

OBJET : désapplication droite d'un champ pile :

. Teste si désapplication possible :

si NOMBCAR (REFCHAMP) - EFFECT (REFCHAMP) ≠ 0 alors termine (REP = - 2)

. Désapplique :

. PTZONE : = REFCHAMP

. PTCHAINE : = COURANT (COPILE)

. LGZ : = NOMBCAR (REFCHAMP)

. émet ADDCH (PTCHAINE, PTZONE, LGZ, CAREN)

. Met à jour le descriptif :

. COURANT (COPILE) : = PTCHAINE

. FIN (COPILE) : = PTCHAINE

. AVANT (COPILE) + = CAREN

. TOTAL + = CAREN

. Emet CHTEST (REFCHAMP, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. termine.

REPONSE

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur.

REP = DESGBA (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ

CONTPAS = 0 si exécution déterministe, 2 sinon

RESULTAT :

DEPART : pointe sur champ suivant.

OBJET : désapplication gauche d'un champ bande :

. Teste si désapplication possible :

si NOMBCAR (REFCHAMP) - EFFECT (REFCHAMP) ≠ 0 alors termine (REP = - 2)

. Désapplique :

. INC : = NOMBCAR (REFCHAMP)

. APRES (COBAND) + = INC

. AVANT (COBAND) - = INC

. COURANT (COBAND) - = INC

. PTZONE : = REFCHAMP

. PTCHAINE : = COURANT (COBAND)

. LGZ : = NOMBCAR (REFCHAMP)

. émet ADDCH (PTCHAINE, CONTPAS, DEPART)

si REP = - 1 termine (REP = - 1)

. Termine.

REPONSE :

REP = 0 fin normale

REP = - 1 DEPART pointe sur champ suivant

REP = - 2 si erreur.

REP = CHTEST (REFCHAMP, CONTPAS, DEPART)

ARGUMENTS D'ACTIVATION :

REFCHAMP = référence du champ ; CONTPAS = 0 si déterministe
2 sinon

RESULTAT :

DEPART = référence du 1er champ différent de la règle suivante.

OBJET : tests de fin de désapplication d'un champ :

. Cas CONTPAS = Q : dépile le champ en émettant : DELPILCH

. Cas CONTPAS = 0 : (exécution non déterministe : teste si il existe un suivant) si SUIV (REFCHAMP) ≠ 0

alors : . DEPART : = REFCHAMP
. termine

sinon : dépile le champ en émettant : DEPILCH

. termine.

REPONSE :

REP = 0 champ dépilé

REP = - 1 si il existe un suivant.

REP = VIDENT (REPONSE)

ARGUMENT RESULTAT = REPONSE = OUI si vide, non sinon

OBJET : teste si l'entrée est vide :

. si APRES (COENTR) = 0 alors REPONSE = OUI
sinon REPONSE = NON

. termine

REPONSE :

REP = 0 si OUI

REP = - 1 si NON.

REP = VIDPIL (REPONSE)

ARGUMENT RESULTAT = REPONSE = OUI si vide, NON sinon

OBJET : Teste si la pile est vide :

. si TOTAL (COPILE) = 0 alors : REPONSE = OUI
sinon : REPONSE = NON

. termine.

REPONSE :

REP = 0 si OUI

REP = - 1 si NON.

REP = MAJET (EDIC)

ARGUMENT D'ACTIVATION :

EDIC = 0 si édition voulue, 2 sinon

OBJET : met à jour l'état courant et l'édite si édition demandée :

. Met à jour :

. COURANT (COETAT) : = DEBUT (COETAT)

. FIN (COETAT) : = DEBUT (COETAT) + TOTAL (COETAT)

. AVANT (COETAT) : = 0

. APRES (COETAT) : = TOTAL (COETAT)

. Edite :

si EDIC = 0 alors :

. MESSAGE : = ETAT :

. MODE : = 0

. REFSTRUC : = REF (COETAT)

. émet : IMPENT (MESSAGE,REFSTRUC , MODE)

. Termine.

REPONSE :

REP = 0 fin normale.

REP = MAJEN (EDIC)

ARGUMENT D'ACTIVATION :

EDIC = 0 si édition voulue, 2 sinon.

OBJET : met à jour l'entrée courante et l'édite si édition demandée :

. Met à jour :

. AVANT (COENTR) : = TOTAL (COENTR) - APRES (COENTR)

. Edite :

si EDIC = 0 alors :

. MESSAGE : = ENTREE :

. MODE : = 2

. REFSTRUC : = REF (COENTR)

. émet : IMPENT (MESSAGE, REFSTRUC, MODE)

. Termine.

REPONSE :

REP = 0 fin normale.

REP = MAJPI (EDIC)

ARGUMENT D'ACTIVATION :

EDIC = 0 si édition voulue, 2 sinon.

OBJET : met à jour la pile courante et l'édite si édition demandée :

. met à jour :

. COURANT (COPILE) : = FIN (COPILE)

. AVANT (COPILE) : = TOTAL (COPILE)

. APRES : = 0

. édite :

si EDIC = 0 alors :

. MESSAGE : = PILE :

. MODE : = 0

. REFSTRUC : = REF (COPILE)

. émet : IMPENT (MESSAGE, REFSTRUC, MODE)

. termine.

REPONSE :

REP = 0 fin normale.

REP = MAJBA (EDIC)

ARGUMENT D'ACTIVATION

EDIC = 0 si édition voulue, 2 sinon.

OBJET : met à jour la bande courante et l'édite si édition demandée :

. Met à jour :

. TOTAL (COBAND) : = AVANT (COBAND) + APRES (COBAND)

. FIN (COBAND) : = COURANT (COBAND) + APRES (COBAND)

. Edite :

si EDIC = 0 alors :

. MESSAGE : = BANDE

. MODE : = 4

. REFSTRUC : = REF (COBAND)

. émet : IMPENT (MESSAGE, REFSTRUC, MODE)

. Termine.

REPONSE :

REP = 0 fin normale.

REP = MAJSO (EDIC)

ARGUMENT D'ACTIVATION

EDIC = 0 si édition voulue, 2 sinon.

OBJET : met à jour la sortie courante et l'édite si édition demandée :

. Met à jour :

. FIN (COSORT) : = COURANT (COSORT)

. AVANT (COSORT) : = TOTAL (COSORT)

. APRES (COSORT) : = 0

. Edite :

si EDIC = 0 alors :

. MESSAGE : = SORTIE

. MODE : = 0

. REFSTRUC : = REP (COSORT)

. émet : IMPENT (MESSAGE, REFSTRUC, MODE)

. Termine.

REPONSE

REP = 0 fin normale.

V-2.7.LA MACHINE ARCHIVAGE

OBJET : gérer l'archivage des entités

MODULES :

MODULE IDENTIFICATION :

OBJET : gère la structure de données associée aux Identificateurs d'entités.

STRUCTURE MANIPULEE : deux listes sont associées aux Identificateurs d'entités :

. d'une part : les identificateurs sont structurés en LISTE selon l'ordre lexicographique des NOMS d'entités.

. d'autre part : pour des raisons de rapidité d'accès, les entités de type classe, ordonnées selon le même ordre, forment une seconde liste.

A ces deux listes se trouvent associés :

. les listes de listes : PREM et CLAS

. le descriptif d'item courant qui repère le courant en liste ainsi que son précédent dans les deux listes.

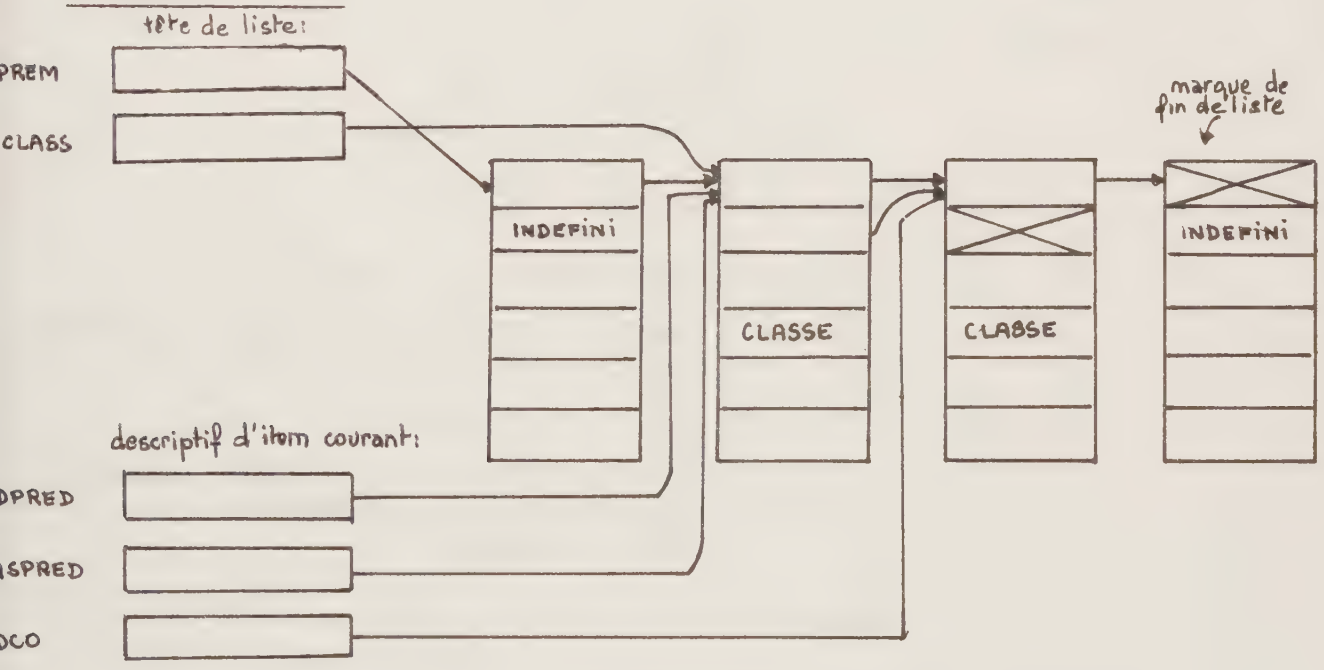
L'élément de liste, c'est-à-dire l'identificateur sera schématisé par une suite de cases consécutives ; chaque case contient un constituant de l'élément. Une case est repérable par le couple :
(< adresse d'élément > , < adresse relative de la case >).

L'association de noms symboliques aux adresses relatives permet l'écriture : NOM SYMB (< adresse d'élément >) pour repérer une case : c'est cette écriture qui sera utilisée dans la suite.

L'identificateur aura donc la structure suivante :

adresse relative	nom symbolique	identificateur	nature du constituant
0	SUIVANT		pointeur vers ident suivant
1	CLASUIV		pointeur vers classe suivante
2	NOM		chaîne de caractères
3	TYPE		chaîne de caractères
4	NOTYPE		entier : code associé au type
5	REF		pointeur vers la structure

Schéma des listes



SOUS MODULES :

SOUS MODULE : GESTION DES LISTES D'IDENTIFICATEURS

OBJET : regroupe les primitives de traitement des listes :

. recherche, création, destruction d'un item, etc...

PRIMITIVES : cf. pages suivantes

REP = INITLIST

pas d'arguments

OBJET : les classes standard (Identificateurs - Descriptifs et Contenus) définies ou constantes sont recopiées dans la zone libre de l'usager et chaînées.

REP = CRELIST (NOM, TYPE, NOTYPE)

OBJET : insert l'item courant dans la liste des identificateurs :

. alloue un identificateur à l'item en émettant :

ALLOC (LGIDENT, REFALLOC)

. si REP = - 1 termine

. remplit l'identificateur :

NOM (REFFALOC) : = ZNOM
TYPE (REFALLOC) : = TYPE
NOTYPE (REFALLOC) : = NOTYPE

. chaîne dans la liste :

. si l'entité est de type classe :

CLASUIV (REFALLOC) : = CLASVIV (CLASPRED)
CLASUIV (CLASPRED) : = REFALLOC

. pour toute entité :

SUIVANT (REFALLOC) : = SUIVANT (ADPRED)
SUIVANT (ADPRED) : = REFALLOC
ADCO : = REFALLOC

. termine

ARGUMENTS D'ACTIVIATION

NOM = nom de l'item en caractères

TYPE = type de l'item en caractères

NOTYPE = code associé au type de l'item.

REPONSE

REP = - 1 allocation impossible : plus de place

REP = 0 fin normale

REP = CHLIST (ZNOM)

OBJET : rend courant dans la liste des identificateurs, celui dont on donne le nom :

- recherche en liste : initialisation :
  - ADPRED pointe sur la tête de liste des ident : PREM
  - CLASPRED pointe sur la tête de liste des classes : CLASS
- recherche en liste : boucle :
  1. passage à l'item suivant :
     - ADCO : = SUIVANT (ADPRED)
  - si ADCO = 0 : fin de liste : termine
  - comparaison des noms : si NOM (ADCO) = ZNOM : termine
  - si NOM (ADCO) > ZNOM : (ordre lexicographique)
    - ADCO : = 0
    - termine
  - progression dans la liste :
    - ADPRED : = ADCO
    - si classe : CLASPRED : = ADCO
    - retourne en 1

ARGUMENT D'ACTIVATION

ZNOM = nom de l'item cherché

REPONSE

REP = - 2 fin de liste
REP = - 1 nom inexistant
REP = 0 nom existant

REP = CHCLAS (ZNOM)

ARGUMENT D'ACTIVATION

ZNOM = nom de la classe cherchée

OBJET : rend courant dans la liste des classes, celle dont on donne le nom :

. recherche en liste : initialisation:

. CLASPRED : = PREM

. recherche en liste : boucle :

1. passage à la classe suivante :

ADCO : = CLASUIV (CLASPRED)

. si ADCO = 0 fin de liste : termine

. comparaison des noms : si NOM (ADCO) = ZNOM : termine

. si NOM (ADCO) ZNOM : (ordre lexicographique)

. ADCO : = 0

. termine

. progression dans la liste :

. CLASPRED = ADCO

. retourne en 1

REPONSE

REP = - 2 fin de liste

REP = - 1 nom existant

REP = 0 nom existant

REP = DETLIST

OBJET = détruit l'item courant dans la liste des identificateurs :

. détruit le chaînage si l'entité est de type classe :

CLASUIV (CLASPRED) : = CLASUIV (ADCO)

. détruit le chaînage entité :

SUIVANT (ADPRED) : = SUIVANT (ADCO)

. remet à zéro le descriptif d'item courant :

ADPRED : = 0

CLASPRED : = 0

ADCO : = 0

REPONSE

REP = 0 fin normale

REP = EDNOTYP

pas d'arguments

OBJET : parcourt la liste des entités et édite le nom et le type de chaque item :

. recherche en liste : initialisation :

ADPRED pointe sur PREM

1. recherche en liste : boucle sur les items :

. passage à l'item suivant :

ADCO : = SUIVANT (ADPRED)

. teste en fin de liste :

si ADCO = 0 alors termine

. édite le nom et le type de l'item courant :

. remplit le tampan :

. ADNOM pointe sur NOM (ADCO)

. ADTYP pointe sur TYPE (ADCO)

. émet : EMPLITAMP (ADNOM,LGNOM)

. émet : AMPLITAMP (ADTYP,LGTYPE)

. édite le tampon : émet

EDITAMP

. progresse dans la liste :

. ADPRED : = ADCO

. retourne en 1

REPONSE :

REP = 0 fin normale

REP = EDNO (TYP)

pas d'arguments

OBJET = parcourt la liste des entités et édite le nom et le type de chaque item :

. recherche en liste : initialisation :

ADPRED pointe sur PREM

1. recherche en liste : boucle sur les items :

. passage à l'item suivant :

ADCO ; = SUIVANT (ADPRED)

. teste si fin de liste :

si ADCO = 0 alors termine

. si TYP (ADCO) = TYP alors édite le nom de l'item courant :

. remplit le tampon :

. ADNOM pointe sur NOM (ADCO)

. émet : EMPLITAMP (ADNOM,LGNOM)

. édite le tampon : émet :

EDITAMP

. progresse dans la liste :

. ADPRED : = ADCO

. retourne en 1

REPONSE =

REP = 0 fin normale

SOUS MODULE : GESTION DES IDENTICATEURS

OBJET : regroupe les primitives de traitement d'un identificateur

PRIMITIVES :

REP = LIDESTRUC (REFSTRUC)

OBJET = Place dans l'identificateur de l'item courant, la référence à la structure qui lui est associée :

- . REFS (ADCO) : = REFSTRUC
- . termine

ARGUMENT D'ACTIVATION

REFSTRUC = référence de la structure associée à l'item courant.

REPONSE =

REP = 0 fin normale

REP = COURANT (REFIDENT,REFSTRUC)

ARGUEMENT RESULTAT

REFIDENT = référence de l'item courant de la liste des identificateurs

REFSTRUC = référence de la structure associée

OBJET = donne l'adresse de l'item courant de la liste des identificateurs s'il existe ; donne également l'adresse de la structure associée

. vérifie qu'il existe un item courant si ADCO = 0 alors :

. REFSTRUC = 0

. termine

. REFIDENT = ADCO

. REFSTRUC = REFS (ADCO)

REPONSE =

REP = 0 fin normale

REP = - 1 si pas d'item courant.

REP = CLASTAND (REPONSE)

ARGUMENT RESULTAT

REPONSE = oui si l'item courant de la liste des identificateurs référence une classe standard, = NON sinon

OBJET = recherche si l'item courant de la liste des identificateurs référence une classe standard :

1. vérifie que l'item est une classe :

si NOTYPE (ADCO) ≠ CODCLAS alors

. REPONSE : = NON

. termine (REP = 1)

2. teste si la classe est standard par consultation du descriptif :

. REFCLAS : = REFS (ADCO)

. émet : STANDARD (REFCLAS, REPONSE)

. termine.

REPONSE :

REP : 0 fin normale

REP = 1 si l'item est une classe

REP = AUTOMATE (REPONSE)

ARGUMENT RESULTAT : REPONSE = OUI si l'item courant est un automate, NON sinon.

OBJET = teste si l'item courant de la liste des identificateurs, s'il existe, est un automate.

. vérifie qu'il existe un item courant : si ADCO = 0 alors termine

. teste si l'item courant est un automate :

si NOTYP (ADCO) = CODAUT alors : REPONSE = OUI

sinon : REPONSE = NON

. termine

REPONSE :

REP = 0 fin normale

REP = - 1 pas d'item courant

REP = CREENT (TYP, NBGENT, REFSTRUC)

ARGUMENTS D'ACTIVATION

TYP = numéro de type de l'entité à créer

NBCENT = longueur de l'entité en caractères de type variable ou file) = 0 sinon.

RESULTAT

RESTRUC = référence du descriptif de l'entité

OBJET = crée une entité :

. si TYP = CODAUT : crée un automate en émettant :

CREAUTO (REFSTRUC) si REP = - 1 termine

. si TYP = CODVAR : crée une variable en émettant :

CREVAR (NBCENT,REFSTRUC) si REP = - 1 termine

. si TYP = CODFIL : crée une file en émettant :

CREFIL (NBCENT, REFSTRUC) si REP = - 1 termine

. si TYP = CODCLAS : crée une classe en émettant :

CRECLAS (REFSTRUC) si REP - 1 termine

. termine

REPONSE

REP = 0 fin normale

REP < 0 erreur lors de la création

REP = ECRIENT (REFDESC)

ARGUMENT D'ACTIVATION :
REFDESC = référence du descriptif de position

OBJET : donne un contenu à l'entité courante :

. si l'entité est un automate : NOTYPE (ADCO) = CODAUTO
alors : . IMPRIME (INCOMPATIBLE)
. termine

. si l'entité est une classe : NOTYPE (ADCO) = CODCLAS
alors . IMPRIME (INCOMPATIBLE)
. termine

. si l'entité est une file : NOTYPE (ADCO) = CODFIL
alors : . vérifie les arguments de position : émet :

CONTROLE (REFDESC, REFSTRUC, PTCHAINE)
si REP : - 1 termine

. donne un contenu à la file en émettant :

ECRIFIL (REFSTRUC, PTCHAINE) si REP : -1 termine

. si l'entité est une variable : NOTYPE = (ADCO) = CODVAR
alors : . donne un contenu à la variable en émettant :

ECRIVAR (REFSTRUC)
si REP = - 1 termine

. termine

REPONSE =

REP = 0 fin normale
REP = - 1 si erreur

REP = COPENT (NOM1)

ARGUMENT D'ACTIVATION

NOM1 = nom de l'entité où l'on doit copier

OBJET = copie l'entité courante dans l'entité repérée par NOM1 :
traite selon le type de l'entité courante :
. si NOTYPE (ADCO) = CODAUT alors :
. REFSTRUC : = REF (ADCO)
. émet : COPIAUT (NOM1, REFSTRUC2)
. si NOTYPE (ADCO) = CODCLAS alors :
. REFSTRUC 2 : = REF (ADCO)
. émet : COPCLAS (NOM1, REFSTRUC2)
si REP = - 1 termine
. si NOTYPE (ADCO) = CODFIL ou CODVAR alors :
. REFSTRUC2 : = REF (ADCO)
. émet : CODVAR (NOM1, REFSTRUC2)
si REP = - 1 termine
. termine

REPONSE

REP = 0 si fin normale
REP = - 1 si erreur

REP = EDITENT

pas d'arguments

OBJET = édite l'item courant de la lsite des identificateurs :

1. cas TYPE (ADCO) = FILE : REFSTRUC : = REF (ADCO)

. traite la référence :

. 1ère position :

. REFDESC = REFDESC1 ;

. émet : CHPOS (REFDESC,TERM)

si REP = - 1 alors termine (REP = - 1)

si REP = 0 alors :

si TERM ≠ , alors termine (REP = 1)

sinon option par défaut

. RES (REFDESC1) : = CODEBUT

. SIGN (REFDESC1) : = +

. INC (REFDESC1) : = 0

2ème position :

. REFDESC = REFDESC2 ;

. émet : CHPOS(REFDESC,TERM)

si REP = - 1 alors : termine (REP = - 1)

si REP = 0 alors : option par défaut :

. RES (REFDESC2) : = CODFIN

. SIGN (REFDESC2) : = -

. INC (REFDESC2) : = 0

. vérifie la compatibilité des arguments en émettant :

REFERENCE (REFDESC1, REFDESC2, REFSTRUC, PCHAINE, LGCHAINE)

. édite la file : en émettant :

IMPCH (PTCHAINE,LGCHAINE)

2. cas TYPE (ADCO) = VARIABLE :

. REFSTRUC : = REF (ADCO)

. PTCHAINE : = DEBUT (REFSTRUC)

. LGCHAINE : = TOTAL (REFSTRUC)

. édite le variable en émettant :

IMPCH (PTCHAINE,LGCHAINE)

3. cas TYPE (ADCO) = CLASSE :

. REFSTRUC : = REP (ADCO)

. édite la classe en émettant :

EDICLAS (REFSTRUC)

4. cas TYPE (ADCO) = AUTOMATE :

. REFSTRUC : = REF (ADCO)

. recherche le paramètre d'édition :

CHNOM (OBLIG,ZNOM) si REP = - 2 termine (REP = - 1)

. vérifie que correct :

TABLE (ZNOM, TPAR, REFITEM) si REP = -1 termine (REP= -1)

. édite selon le paramètre :

si REFITEM = 0 alors : émet FORME ;

sinon : émet : EDITAUTO (ADCO,REFSTRUC)

5. termine

REPONSE

REP = 0 fin normale

REP = - 1 si erreur

MODULE : CONTENU :

OBJET : gère les structures de données utilisées pour archiver les contenus des entités.

SOUS-MODULES :

SOUS-MODULE : GESTION DES AUTOMATES :

OBJET : gère la structure associée aux automates.

STRUCTURES MANIUPULEES :

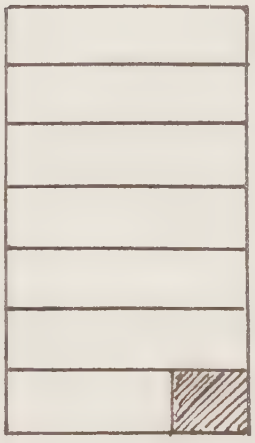
1. le DESCRIPTIF D'AUTOMATE : il s'agit d'un tableau contenant les caractéristiques de l'automate considéré :

Nom Symbolique		Descriptif	Nature du constituant
NBGAUCHE			nombre de champs gauche dans une règle
NBDROIT			nombre de champs droit dans une règle
NBOPERAT			nombre d' opérateurs dans une règle
TYPEOPERA	[0]		tableau des opérateurs d' une règle
	[1]		
	[2]		
	[3]		
	[4]		
	[5]		
	[6]		
	[7]		
	[8]		
REGLES			lien avec l'ensemble des règles

2. l'arborescence des règles : pour ranger les règles, on a choisi une structure d'arborescence afin d'optimiser le temps de recherche lors de l'activation des automates. A chaque règle correspond la liste de ses CHAMPS dans l'ordre naturel.

   Mais lorsque les règles ont leurs premiers champs identiques, ces champs sont mis en commun. Les champs sont classés par ordre lexical.

   Un descriptif de champ se présente sous la forme suivante :

NOM symbolique	Descriptif	Nature du constituant
SUIV		référence du champ suivant dans la liste des champs
SUCC		référence du champ successeur dans la règle
NOMBCAR		nombre de caractères dans le champ
EFFECT		nombre effectif de caractères (sans les blancs)
ZONE		chaine de caracteres constituant le champ
		cadres à gauche

REP = CREAUTO (REFSTRUC)

OBJET = crée une entité de type AUTOMATE :

. alloue un descriptif d'automate : émet :

ALLOC (LGDAUTO, REFSTRUC) si REP = -1 termine

. remet à zéro le descriptif alloué : émet :

RAZ (REFSTRUC,LGDAUTO)

. remplit la table des opérateurs du descriptif en fonction des choix de l'usager :

1 . initialisation :

. émet : IMPRIME (CHOIX DES OPERATEURS)

. positionne le pointeur PTDESC en début de table :

PTDESC : = TYPEOPERA (REFSTRUC)

. demande si un ETAT est désiré : émet :

OUINON (ETAT,REPONSE)

. si REP = -1 : on détient 1'item en cours de création

. REFSTRUC : = 0

. émet : DESALLOC (LGDAUTO)

. termine

. si REPONSE = OUI = met à jour le descriptif :

. NG (REFSTRUC) + = 1

. ND (REFSTRUC) + = 1

. NBOPERAT (REFSTRUC) + = 1

. place le code de 1'opérateur dans la table

TYPEOPERA (PTDESC) : = CODETAT

. incrémente le pointeur : PTDESC + = 1

. demande si une ENTREE est désirée : émet :

OUINON (ENTREE,REPONSE)

. si REP = - 1 : on détruit l'item en cours de création

. REFSTRUC : = 0

. émet : DESALLOC (LGDAUTO)

. termine

. si REPONSE = OUI : met à jour le descriptif :

. NG (REFSTRUC) + = 1

. NBOPERAT (REFSTRUC) + = 1

. place le code de l'opérateur dans la table :
TYPEOPERA (PTDESC) : = CODENTR

. incrémente le pointeur : PTDESC + = 1

. demande si une AVANCE - ENTREE est désirée : émet :
OUINON (AVANCE ENTREE,REPONSE)

. si REP =- 1 : destruction :

. REFSTRUC : = 0

. émet : DESALLOC (LGAUTO)

. termine

. si REPONSE = oui : met à jour :

. ND (REFSTRUC) + = 1

. NBOPERAT (REFSTRUC) + = 1

. place le code dans la table :
TYPEOPERA (PTDESC) : = CODAVE

. PTDESC + = 1.

. demande si une PILE est désirée : émet :
OUINON (PILE,REPONSE)

. si REP = -1 : on détruit l'item en cours de création =

. RESTRUC : = 0

. émet : DESALLOC (LGDAUTO)

. termine

. si REPONSE = OUI : met à jour le descriptif :

. NG(REFSTRUC) + = 1

. ND(REFSTRUC) + = 1

. NBOPERAT (REFSTRUC) + = 1

. place le code dans la table :

TYPEOPERA (PTDESC) : = CODPILE

. incrémente le pointeur : PTDESC + = 1

. demande si une BANDE est désirée émet :

OUINON (BANDE, REPONSE)

. si REP = - 1 : on détruit l'item en cours de création

. REFSTRUC : = 0

. émet : DESALLOC (LGAUTO)

. termine.

. si REPONSE = OUI : met à jour le descriptif :

. NG (REFSTRUC) + = 1

. ND (REFSTRUC) + = 1

. NBOPERAT (REFSTRUC) + = 1

. place le code de 1'opérateur dans la table :

TYPEOPERA (PTDESC) : = CODBANDE

. incrémente le pointeur : PTDESC + = 1

. demande si une AVANCE BANDE est désirée :

OUI NON (AVANCE BANDE,REPONSE)

. si REP = - 1 : destruction :

. REFSTRUC : = 0

. émet : DESALLOC (LGDAUTO)

. termine

. si REPONSE = OUI : met à jour :

. ND (REFSTRUC) + = 1

. NBOPERAT (REFSTRUC) + - 1

. place le code dans la table :

TYPEOPERA (PTDESC) : = CODAVB

. PTDESC + = 1

. demande si une SORTIE est désirée : émet :

OUINON (SORTIE,REPONSE)

. si REP = - 1 : on détruit 1'item en cours de création :

. REFSTRUC : = 0

. émet : DESALLOC (LGDAUTO)

. termine

. si REPONSE = OUI : met à jour le descriptif :

. ND (REFSTRUC) + = 1

. NBOPERAT (REFSTRUC) + = 1

. place le code dans la table :

TYPEOPERA (PTDESC) : = CODSOR

. incrémente le pointeur : PTDESC + = 1

. demande si une AVANCE SORTIE est désirée : émet :

OUINON (AVZNCE SORTIE,REPONSE)

. si REP = - 1 : destruction

. REFSTRUC = 0

. émet DESALLOC (LGDAUTO)

. termine

. si REPONSE = OUI : met à jour :

. ND (REFSTRUC) + = 1

. NBOPERAT (REFSTRUC) + = 1

. place le code dans la table :

TYPEOPERA (PTDESC) : = CODANS

. PTDESC + = 1

. vérifie que le nombre de champs gauche et le nombre de champ droit sont différents de zéro :

. si ND (REFSTRUC) = 0 et/ou NG(REFSTRUC) = 0 :

. émet IMPRIME (INCOMPATIBLE)

. retourne en 1 pour nouveau choix.

. visualise la forme des règles en émettant :

FORME (REFSTRUC)

. termine.

ARGUMENT RESULTAT : REFSTRUC : référence du descriptif de l'entité.

REPONSE

REP = - 2 arrêt sur erreur usager : destruction de l'item
REP = - 1 allocation impossible : plus de place
REP = 0 si fin normale

REP = FORME (REFSTRUC)

ARGUMENT D'ACTIVATION

REFSTRUC = référence du descriptif de l'automate à traiter.

OBJET : visualise la forme des règles d'un automate

1. remplit le tampon :

. traite le premier membre

. initialise :

. PTDESC = TYPEOPERA (REFSTRUC) ; NBOP : = 0 ;

. émet INITAMP pour initialiser le tampon d'E/S.

a) . traite par cas selon le type : tant que NBOP ≤ NBOPERA (REFSTRUC) faire :

. cas état : si TYPEOPERA (PTDESC) = CODETAT alors :
place l'opérateur dans le tampon en émettant :

EMPLITAMP (TFORM (CODETAT), LGOP)

. cas entrée : si TYPEOPERA (PTDESC) = CODENT alors :
place l'opérateur dans le tampon en émettant :

EMPITAMP (TFORM (CODENT), LGOP)

. cas pile : si TYPEOPERA (PTDESC) = CODPIL alors :
place l'opérateur dans le tampon en émettant :

EMPLITAMP (TFORM (CODPIL), LGOP)

. cas bande : si TYPEOPERA (PTDESC) = CODBA alors :
place l'opérateur dans le tampon en émettant :
EMPLITAMP (TFORM(CODBA),LGOP)

. place le séparateur : si TYPEOPERA (PTDESC) = CODBA
alors : TERM : = '—> ' (fin de membre)
sinon : TERM : = ','
émet EMPLITAMP (TERM,LGTERM)

. incrémente pointeur et compteur :

PTDESC + = 1 ; NBOP + = 1 ;

. retourne en a.

. positionne un pointeur dans le tampon (pour édition en deux fois si nécessaire) en émettant :

SETPTAMP

. traite le second membre :

. initialise :

PTDESC : = TYPEOPERA (REFSTRUC)

NBOP : = 0

b. traite par cas selon le type : tant que :

NBOP ≤ NBOPERA (REFSTRUC) faire :

. cas état : si TYPEOPERA (PTDESC) = CODETAT alors : place l'opérateur dans le tampon en émettant :

EMPLITAMP (TFORM (CODENT), LGOP)

. cas avance-entrée : si TYPEOPERA (PTDESC) = CODAVE alors : place l'opérateur dans le tampon en émettant =

EMPLITAMP (TFORM (CODAVE), LGOP)

. cas bande : si TYPEOPERA (PTDESC) = CODBA alors :

place l'opérateur dans le tampon en émettant:

EMPLITAMP (TFORM (CODBA), LGOP)

. cas avance-bande : si TYPEOPERA (PTDESC) : CODAVBA place l'opérateur dans le tampon en émettant :

EMPLITAMP (TFORM(CODAVBA), LGOP)

. cas sortie : si TYPEOPERA (PTDESC) = CODSOR place l'opérateur dans le tampon en émettant =

EMPLITAMP (TFORM (CODSOR) , LGOP)

. cas avance-sortie : si TYPEOPERA (PTDESC) = CODAVSO place l'opérateur dans le tampon en émettant :

EMPLITAMP (TPFORM) CODANSO), LGOP)

. place le séparateur si TYPEOPERA (PTDESC) = CODAVSO alors : TERM = '.' (fin de règle)

sinon : TERM = ','

émet EMPLITAMP (TERM,LGTERM)

. incrémente pointeur et compteur :

PTDESC + = 1 ; NBOP + = 1

retourne en b

2. édite le tampon ainsi rempli : en émettant:

EDITAMP

3. termine

REPONSE = REP = 0 fin normale

REP = EDITAUTO (ADCO,REFSTRUC)

ARGUMENTS D'ACTIVATION

ADCO = référence de l'identificateur de l'automate courant.
REFSTRUC = référence du descriptif de l'automate courant.

OBJET : édite l'automate courant :

. place les séparateurs en émettant :

SEPAR(ADCO,REP,SAVEM,NBCHAMP)

. initialise le parcourt de l'arborescence :

. PTSEP pointe sur REP (0)
. PTARB pointe sur REGLES (ADCO)
. NCP pointe sur ADZ (0)

1.parcourt l'arborescence :

. PTARB : = Succ. (PTARB)
. si PTARB ≠ 0 alors

2. place le champ dans le tampon :

. LGITEM : = NOMBCAR (PTARB)
. REFITEM : = ZONE (PTARB)
. émet : EMPLITAMP (LGTIM,REFITEM)

. place le séparateur :

. émet : EMPLITAMP (CGCAR, PTSEP)
. incrémente PTSEP : PTSEP + = 1
. retourne en 1.

sinon :

. édite le tampon émet : EDITAMP

3. recherche le suivant :

. NCP = NCP - 1 si NCP < 0 termine

. PTSEP = PTSEP - 1

. dépile le dernier dans REPONS :

. LG = NOMBCAR (PTARB) + LGAR

. émet PTREP (LG)

. PTARB : = ADZ (NCP)

. PTARB : = SUIV (PTARB)

. si PTARB = 0 alors retourne en 3)
sinon : retourne en 2)

REPONSE :

REP = 0 fin normale

REP = INITREGL (REFSTRUC)

ARGUMENT D'ACTIVATION

REFSTRUC = référence du descriptif de l'automate courant.

OBJET : initialise les variables du module de gestion des règles :

NG = NG (REFSTRUC),

ND = ND (REFSTRUC),
NBCHAMP : = NG + ND

DESC : = REFSTRUC

. termine

REPONSE =

REP = 0 fin normale

REP = LIREGLE (REFALLOC)

ARG. RESULTAT REFALLOC = Ad. de construction de la règle

OBJET = lit et normalise une règle :

. alloue une pile qui contiendra les champs en émettant :

ALLOC (LGREGL, REFALLOC)

si REP = - 1 termine.

a. lit la règle en émettant :

LECT (REGLE :)

si REP = 0 termine

. boucle sur les champs :

. initialisation :

. met à zéro la pile de description de la règle en émettant :

RAZ (PILREG, LGPILREG)

. initialise le pointeur de la file des champs :

ADCONS : = REFALLOC

. initialise la cellule qui contiendra le terminateur de chaque champ : TERM : = ,

. traitement 1er membre :

. positionne les variables :

NORM = OUI (pas de champ avance dans le premier membre)

NUMCP = NG

POUS = 1 (car 1er membre)

. initialise la pile de description :

PTPILE : = PILREG

PILREG (PTPILE) : = ADCONS

b . tant que NUMCP > 0 faire :

. traite un champ en émettant :

CHSUCC (TERM, ADCONS, NORM, POUS, MOTSUIV)

. place le champ dans la pile de description

PTPILE : = PTPILE + 1

PILREG (PTPILE) : = MOTSUIV

. incrémente pointeurs et COMPteurs :

NUMCP - = 1 ; ADCONS : = MOTSUIV ;

. retourne en b.

si TERM ≠ > alors : émet ZER (POUS) ;

sinon : TERM : = ,

POUS : = 2.

c. traitement 2ème membre :

. initialise les variables :

NBOP = 0

PTDESC : = TYPEOPERA (DESC)

d. tant que : NBOP ≤ NBOPERA (DESC) faire

. cas état : si TYPEOPERA (PTDESC) = CODETAT alors : NORM = OUI

. cas avance-entrée : si TYPEOPERA (PTDESC) = CODAVE alors : NORM = NON

. cas pile : si TYPEOPERA (PTDESC) = CODPIL alors : NORM = OUI

. cas bande : si TYPEOPERA (PTDESC) = CODBA alors : NORM : OUI

. cas avance-bande : si TYPEOPERA (PTDESC) = CODAVB alors : NORM = NON

. cas sortie : si TYPEOPERA (PTDESC) = CONDSOR alors : NORM = OUI

. cas avance-sortie : si TYPEOPERA (PTDESC) = CODAVSO alors : NORM = NON

. émet CHSUCC (TERM, ADCONS, NORM, POUS, MOTSUIV)

. place le champ dans la pile de description :

. PTPILE + = 1

. PILREG (PTPILE) : = MOTSUIV

. incrémente pointeurs et compteurs:

. NBOP - = 1

. PTDESC + = 1

. ADCONS : = MOTSUIV

. retourne en d.

. si TERM ≠ LF alors :

. émet ZER (POUS)

. si POUS = 1 retourne en a. sinon retourne en c.

. désalloue la place non utilisée:

. calcule la longueur non utilisée : LGNUTI en mots :
longueur non utilisée = lg allouée - lg utilisée

LGNUTI = LGREGL - (MOTSUIV-REFALLOC)

. désalloue en émettant.
DESALLOC (LGNUTI)

. termine

REPONSE =

REP = 0 si fin normale

REP = - 1 si erreur

REP = CHSUCC (TERM, ADCONS, NORM, POUS, MOTSUIV)

ARGUMENTS D'ACTIVATION :

TERM = terminateur du champ précédent

ADCONS = pointeur sur début de champ

NORM = OUI si pas de champ avance, NON sinon

POUS = 1 si 1er membre, 2 si 2nd membre

ARGUMENT RESULTAT :

MOTSUIV = 1er cellule libre dans la pile des champs.

OBJET : normalise un champ :

. teste le terminateur précédent : si TERM ≠ , alors : émet
ZER (POUS)

. alloue un champ dans la pile des champs :

. met à zéro le descriptif de champ : en émettant :

RAZ (ADCONS, LGINBC)

. initialise un pointeur en début de zone du champ :

PTZONE = ADCONS + LGINBC

. normalise un champ :

. initialisation : TRCLAS = NON (pas de classe en cours de traitement)

a). normalise un champ en émettant :

ZONE (ADCONS, PTZONE, TRCLAS, TERM, MOTSUIV)

. si TERM = LF : le champ est "à cheval" sur deux lignes TTY:
lit la ligne suivante en émettant :

LECT (BLANC)

. si REP = 0 (réponse vide) : TERM = LF
. si REP > 0 suite de la normalisation : retourne en a ;

. traite les champs avance : (si NORM = NON) :

si NBCAR (ADCONS) > 0 alors :

. ENTIER : = ZONE (ADCONS)
. convertit ENTIER en binaire en émettant :

NOMBRE (ENTIER, NBCAR (ADCONS))

si REP = - 1 émet ZER (POUS)

. si l'avance est trop importante :

MAXAVP < ENTIER ou MAXAVN > ENTIER

émet ZER (POUS)

. place le nombre en binaire dans le descriptif de champ

EFFECT (ADCONS) : = ENTIER

. termine

REPONSE

REP = 0 fin normale.

REP = ZER (POUS)

ARGUMENT D'ACTIVATION :

POUS = 1 si 1er membre, 2 si 2nd

OBJET : module d'erreur des primitives LIREGLE et CHSUCC, est activé sur erreur :

. si POUS = 1 alors : . émet : IMPRIME (ERREUR PREMIER MEMBRE)

. retourne

sinon : . émet : IMPRIME (ERREUR SECOND MEMBRE)

. lit le second membre : émet :

LECT (- >)

. réinitialise avant traitement :

TERM = ?

ADCONS : = PILREG (NG + 1)

. retourne

Pas de REPONSE

REP = ZONE (ADCONS, PTZONE, TRCLAS, TERM, MOTSUIV)

ARGUMENTS D'ACTIVATION :

ADCONS = pointeur sur début de champ

PTZONE = pointeur sur zone du champ

TRCLAS = OUI si classe en cours de traitement, NON sinon.

ARGUMENTS RESULTAT :

PTZONE = nouveau pointeur sur zone

TERM = terminateur rencontré

MOTSUIV = 1er mot libre de la pile des champs

OBJET : normalisation d'un champ (si le champ est "à cheval" sur deux lignes deux appels seront nécessaires)

a) . lit un caractère émettant :

CARTAMP (CAR)

si REP = -1 : b) . TERM : = LF

c) . émet MASCAR (PTZONE)

. repère la 1ère cellule vide :

. si REP = 0 MOTSUIV : = PTZONE

. si REP > 0 MOTZUIV : = PTZONE + 1.

. termine

d) . si pas de classe en cours en traitement (TRCLAS = NON) : teste le caractère obtenu :

. si CAR =>: . TERM = >

. retourne en b)

. si CAR = -:. sauvegarde le caractère : SAVE:=CAR

. lit le caractère suivant : émet :

CARTAMP (CAR)

. si REP = - 1 : retourne en b)

. si CAR = > : . TERM : =

. retourne en c)

. place le caractère - dans le champ :

CARCHAM (SAVE, ADCONS, TRCLAS)

. retourne en d)

. place le caractère dans le champ en émettant :

CARCHAM (SAVE, ADCONS, TRCLAS)

. retour en a)

REPONSE

REP = 0 fin normale.

REP = CARCHAM (CAR, ADCONS, TRCLAS)

ARGUMENTS D'ACTIVATION

CAR = caractère à placer

ADCONS = Ad du descriptif du champ traité

TRCLAS = OUI si classe en cours de traitement, NON sinon

ARGUMENT RESULTAT

TRCLAS = OUI si classe, NON sinon

OBJET : place un caractère dans la zone d'un champ et modifie son descriptif en conséquence :

. place le caractère dans le byte repéré par PTZONE

. incrémente PTZONE

. incrémente NOMBCAR (ADCONS)

. teste si incrémentation d'EFFECT nécessaire :

si TRCLAS = OUI alors : si CAR = alors : . TRCLAS := NON

. EFFECT(ADCONS) + = 1

Sinon : si CAR = alors : TRCLAS = OUI

sinon : EFFECT(ADCONS) +=1

. termine

REPONSE

REP = 0 fin normale

REP = CHREGLE (PTPILE, PTPREC, NUMCP, PTAUT)

ARGUMENT RESULTAT

PTPILE = pointeur dans PILREG sur 1er champ différent

PTPREC = référence du successeur du dernier champ trouvé

NUMCP = N° du dernier champ différent

PTAUP = sommet de la pile PILAUT contenant les pointeurs sur prédécesseurs des champs de la règle.

OBJET : cherche si la règle lue (et normalisée par LIREGLE) existe déjà dans l'arborescence de l'automate courant :

. initialise avant recherche :

. initialise le pointeur de la pile de description de la règle lue :

PTPILE pointe sur PILREG

. initialise le pointeur de la pile de description des règles de l'arborescence :

PTAUT pointe sur PILAUT

. recherche la règle :

. initialise avant boucle sur les champs :

. le compteur de champ : NUMCP = 0

. le pointeur dans l'arborescence :

PTARB pointe sur REGLES (DESC)

a). Boucle sur les champs successeurs :

. progresse dans l'arborescence : PTARB = SUCC (PTARB)

. si NUMCP $\leqslant$ NBCHAMP alors :

. recherche dans la liste des champs SUIVANT émet : CHZONE (PTARB, PTZONE, PTPREC)

si REP $\neq$ 0 (champ trouvé) alors :

. sauvegarde l'adresse du champ précédent

PILAUT (PTAUT) : = PTPREC

. incrémente

PTAUT + = 1

PTPILE + = 1

NUMCP + = 1

. retourne en a)

sinon termine (REP = 0)

. termine (REP $>$ 0)

REPONSE

REP $>$ 0 règle déjà existante

REP = 0 règle pas complètement trouvée.

REP = CHZONE (PTARB, PTZONE, PTPREC)

ARGUMENTS D'ACTIVATION

PTARB = référence du pointeur sur champ suivant

PTZONE = pointeur sur zone du champ de la règle lue

RESULTAT

PTARB = référence du pointeur sur champ suivant

PTPREC = référence du champ précédent.

OBJET : recherche dans la liste des suivants d'un champ extraite de l'arborescence des règles si un champ est identique à celui repéré par PTZONE :

. initialise avant boucle :

PTREC = PTARB

. boucle sur les champs suivants :

a) . passage au champ suivant

PTARB : = SUIVANT (PTARB)

. si PTARB = 0 alors termine.

. effectue la comparaison champ-zone :

. calcule la longueur du champ de la liste en mot :

LGMCHA : = NOMBCAR (PTARB) /4

. compare mot par mot

. si pas identique alors : si champ liste > champ dans l'ordre lexicographique alors :

. PTARB : = 0
. termine
sinon
. PTPREC : = PTARB
. retourne en a)

. termine

REPONSE :

REP = 0 champ pas trouvé

REP > 0 champ trouvé

REP = AJOUREGLE (REFALLOC, PTPREC, PTPILE, NUMCP)

ARGUMENTS D'ACTIVATION :

REFALLOC = Adresse de construction de la règle

PTPREC = référence du successeur du dernier champ trouvé

PTPILE = pointeur dans PILREG sur 1er champ différent.

OBJET : chaine dans l'arborescence des règles les champs non trouvés par CHREGLE :

1) chaine le premier champ non trouvé dans la liste des suivants

. NUMCP - = 1

. Mise à zéro des liens en émettant : RAZ (REFALLOC, LGLIENCH)

. chainage suivant : SUIV (REFALLOC) : = SUIV (PTPREC)

SUIV (PTPREC) : = REFALLOC

. calcule la longueur en mots de la zone :

LGMCHA : = NBCAR (PILREG (PTPILE)/4

. écrase des champs déjà existant: déplacement de la zone repérée par PILREG (PTPILE)

. incrémente : PTPILE + = 1

PTPREC : = REFALLOC

. calcule l'adresse de création du prochain champ

REFALLOC : = REFALLOC + LGMCHA + LGIENCH

2) chaine les champs successeurs :

a) . tant que NUMCP $\leqslant$ NBCHAMP faire :

. émet CHAINSUCC (PTPREC, REFALLOC, PTPILE) pour chainer

. incrémente : PTPILE + = 1

NUMCP - = 1

. retourne en a)

. met à zéro du successeur du dernier champ

SUCC (PTPREC) : = 0

3) récupère la place non utilisée

. sauvegarde : SAVE : = REFALLOC

. demande l'état du pointeur en émettant :

ALLOC (0, REFALLOC)

. calcule la longueur à désallouer : LG = REFALLOC - SAVE

. désalloue la longueur non utilisée : émet :

DESALLOC (LG)

4) termine

REPONSE

REP = 0 fin normale.

REP = CHAINSUCC (PTPREC, REFALLOC)

ARGUMENT D'ACTIVATION

REFALLOC : = Adresse de construction du champ

PTPREC : = référence du champ précédent

RESULTAT

PTPREC : = référence du champ précédent.

OBJET : chaine un champ successeur :

. chainage champ précédent - champ courant :

SUCC (PTPREC) = REFALLOC

. calcul de la longueur en mots de la zone :

LGMCHA = NOMBCAR (PILREG (PTPILE) / 4

. déplace la zone repérée par PILREG (PTPILE)

. sauvegarde : PTPREC : = REFALLOC

. calcule l'adresse de création du champ successeur :

REFALLOC : = REFALLOC + LGLIENCH + LGMCHA

. termine.

REP = OTREGLE (PTAUT, NUMCP)

ARGUMENT D'ACTIVATION

PTAUT = sommet de la pile contenant les pointeurs sur prédecesseurs des champs de la règle.

NUMCP = N° du dernier champ.

OBJET : ôte la règle lue : boucle sur les champs

tant que NUMCP > 0 faire :

. décrémente compteur et pointeur

. NUMCP - = 1
. PTAUT - = 1

. calcule l'adresse du précédent :

PTPREC : = PILAUT (PTAUT)

. Si SUCC (SUIVANT (PTPREC)) ≠ 0 alors, termine.

Sinon : . détruit le champ

NOMBCAR (SUIVANT (PTPREC)) : = - 1

. établit le nouveau chaînage :

SUIVANT (PTPREC) : = SUIVANT (SUIVANT (PTPREC))

. termine

REPONSE

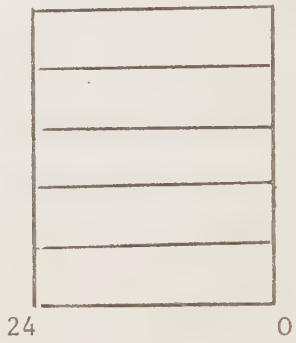
REP = 0 fin normale.

SOUS MODULE : GESTION DES CLASSES

OBJET : gère la structure associée aux classes de caractères.

STRUCTURE MANIPULEE : LE DESCRIPTIF DE CLASSE : une classe est représentée par un tableau booléen ; la structure d'un descriptif de classe est la suivante :

Nom symbolique :		Descriptif	Nature du constituant
DEBUT			Pointeur sur CHAINCAR
STANDARD			Entier : 0 si classe non standard 2 sinon
CHAINCAR	(0)		Le bit n° J du mot CHAINCAR (I)
	(1)		représente le caractère codé :
	(2)		24 * I + J en machine



REP = CRECLAS (REFSTRUC)

OBJET : Crée une entité de type CLASSE et l'initialise à vide :

. alloue un descriptif de classe émet :

ALLOC (LGDCLA, REFSTRUC) si REP = - 1 termine.

. alloue le contenu : émet :

ALLOC (LGCLA, REFCONT) si REP = - 1 termine.

. initialise le descriptif : émet :

DEBUT (REFSTRUC) : = REFCONT

STANDARD (REFSTRUC) : = 0

. initialise le contenu : émet :

RAZ (REFCONT, LDCLA)

ARGUMENT RESULTAT :

REFSTRUC = référence du descriptif de l'entité.

REPONSE :

REP = - 1 : allocation impossible : plus de place

REP = 0 : si fin normale.

REP = STANDARD (REFCLAS, REPONSE)

ARGUMENTS D'ACTIVATION

REFCLAS = référence du descriptif de la classe à traiter

RESULTAT

REPONSE = OUI si la classe est standard, NON sinon

OBJET : recherche si une classe est standard

. si STANDARD (REFCLAS) = 0 alors :

. REPONSE : = NON
. termine

sinon

. REPONSE : = OUI

. termine

REPONSE

REP = 0 fin normale

REP = COPCLAS (NOM1, REFSTRUC2)

ARGUMENTS D'ACTIVATION :

NOM1 = nom de l'entité où l'on copie

REFSTRUC2 = référence du descriptif de l'entité à copier.

OBJET : copie une classe dans une autre :

. rend courante l'entité repérée par NOM1 en émettant

CHLIST (OBLIG, NOM1)

. vérifie que l'entité courante est une classe non standard en émettant

CLASTAND (REPONSE) si REP = 0 : . émet IMPRIME (INTERDIT)

. termine

. calcule l'adresse du descriptif de l'entité courante

REFSTRUC1 : = REF (ADCO)

. copie dans la classe rérérencée par REFSTRUC1 la classe référencée par REFSTRUC2 ;

CHAINCAR (REFSTRUC1) : = CHAINCAR (REFSTRUC2)

CHAINCAR (REFSTRUC1+ 1) : = CHAINCAR (REFSTRUC2 + 1)

CHAINCAR (REFSTRUC1 + 2) : = CHAINCAR (REFSTRUC2 + 2)

. termine

REPONSE

REP = 0 fin normale

REP = - 1 si erreur

REP = CHCAR (REFSTRUC, CAR, NOMOT, RESTE, CARES)

ARGUMENTS D'ACTIVATION :

REFSTRUC = référence du descriptif de la classe.

CAR = caractère.

RESULTAT :

NOMOT = N° du mot où figure le caractère

RESTE = N° du bit dans le mot

CARES = 0 si le caractère n'appartient pas à la classe, ≠ sinon.

OBJET : recherche si le caractère qu'on lui communique appartient à la classe courante :

. positionne 1 pointeur sur le tableau booléen :
PTCAR = DEBUT (REFSTRUC)

. calcule l'adresse relative du caractère dans le tableau :
si CAR - 24 ≤ 0 alors : . NOMOT : = 0
. RESTE : = CAR
sinon : . NOMOT : = QUOTIENT (CAR/24)
. RESTE : = RESTE (CAR/24)

. calcule les arguments résultat :
. CARES = CHAINCAR (PTCAR + NOMOT)
. écrase la cellule en ne gardant que le bit concerné
CARES = TABMASK (RESTE) ET CARES

. termine.

REPONSE

REP = 0 fin normale.

REP = AJCAR (REFSTRUC, NOMOT, RESTE)

ARGUMENTS D'ACTIVATION :

REFSTRUC = référence du descriptif de la classe

NOMOT = N° du mot où figure le caractère

RESTE = N° du bit dans le mot.

OBJET : ajoute le caractère à la classe :

. calcule l'adresse du tableau :

PTCAR : = DEBUT (REFSTRUC)

. CHAINCAR (PTCAR + NOMOT) : = CHAINCAR (PTCAR + NOMOT) OU TABMASK (RESTE)

. termine.

REPONSE

REP = 0 fin normale.

REP = OTCAR (REFSTRUC, NOMOT, RESTE)

ARGUMENTS D'ACTIVATION :

REFSTRUC = référence du descriptif de la classe

NOMOT : N° du mot où figure le caractère

RESTE = N° du bit dans le mot.

OBJET : ôte le caractère à la classe :

. calcule l'adresse du tableau :

PTCAR : = DEBUT (REFSTRUC)

. CHAINCAR (PTCAR + NOMOT)

: = CHAINCAR (PTCAR + NOMOT) NAND TABMASK (RESTE)

. termine

REPONSE : REP = 0 fin normale.

REP = EDICLAS (REFSTRUC)

ARGUMENT D'ACTIVATION :

REFSTRUC : = référence du descriptif de la classe

OBJET : édite une classe :

. boucle sur les mots : Initialisation :

. ADMOT : = DEBUT (REFSTRUC)

. INC : = 0 ; MOT : = 0

2). boucle sur les mots : tant que MOT < LGCLAS faire :

. boucle sur les caractères : Initialisation :

NBCMOT : = 0

1). boucle sur les caractères : tant que NBCMOT < 24 faire

. CAR : = CHAINCAR (ADMOT)

. CAR : = CAR ET TABMASK (NBCMOT)

. si CAR ≠ 0 alors : . CAR = INC + NBMOT

. place le caractère dans REPONS

. NBCMOT + = 1

. retourne en 1)

.Incrémente :

. INC + = 24 (24 bits par mot)

. ADMOT + = 1

. retourne en 2)

. Edite le tampon : émet

EDITAMP

. termine.

REPONSE

REP = 0 si fin normale.

SOUS MODULE : GESTION DES FILES ET DES VARIABLES

OBJET : gère la structure associée aux files et aux variables ;

STRUCTURE MANIPULEE : LE DESCRIPTIF DE CHAINE : la structure est la suivante :

Nom symbolique	Descriptif	Nature du Constituant
NBCMAX		nombre de caractères maximum dans la chaine
TOTAL		nombre de caractères existants dans la chaine
DEBUT		pointeur sur debut de chaîne
COURANT		pointeur courant dans chaîne
FIN		pointeur sur fin de chaîne
AVANT		nombre de caracteres avant le pointeur COURANT
APRES		nombre de caracteres entre COURANT et FIN
CHAÎNE		chaine de caracteres cadres à gauche

REP : CREVAR (NBLENT, REFSTRUC)

OBJET : crée une entité de type VARIABLE et l'initialise à vide :

. alloue un descriptif de variable : émet :

ALLOC (LGDVAR, REFSTRUC) si REP = - 1 termine

. calcule la longueur en mots de la chaine pour allocation :

. si la longueur n'a pas été précisée (NBCENT = 0) prend :

NBCENT : = 16 par défaut.

. LGENT : = CEIL [NBCENT/4] (4 caractères par mot)

. alloue le contenu : émet :

ALLOC (LGENT, REFCONT) si REP = - 1 termine

. initialise le descriptif :

NBCMAX (REFSTRUC) : = NBCENT

TOTAL (REFSTRUC) : = 0

DEBUT (REFSTRUC) : = REFCONT

COURANT (REFSTRUC) : = REFCONT

FIN (REFSTRUC) : = REFCONT

AVANT (REFSTRUC) : = 0

APRES (REFSTRUC) : = 0

. initialise à vide le contenu : émet :

RAZ (REFCONT, LGENT)

ARGUMENT D'ACTIVATION :

NBCENT = longueur de la variable en caractères, ou 0 si pas précisée

RESULTAT :

REFSTRUC = référence du descriptif de l'entité.

REPONSE :

REP = - 1 : allocation impossible : plus de place

REP = 0 : si fin normale.

REP = ECRIVAR (REFSTRUC)

ARGUMENT D'ACTIVATION

REFSTRUC : référence du descriptif de la variable.

OBJET : donne un contenu à une variable d'état :

. initialise le descriptif :

. TOTAL (REFSTRUC) : = 0

. COURANT (REFSTRUC) : = DEBUT (REFSTRUC)

. APRES (REFSTRUC) : = 0

. AVANT (REFSTRUC) : = 0

. donne un contenu :

. PTCHAINE : = DEBUT (REFSTRUC)

. AVANT : = AVANT (REFSTRUC)

. NBCMAX : = NBCMAX (REFSTRUC)

. émet CHAINE (PTCHAINE, AVANT, NBCMAX, NBCAR)

si REP = - 1 termine

. met à jour le descriptif :

. COURANT (REFSTRUC) : = PTCHAINE

. FIN (REFSTRUC) : = PTCHAINE

. AVANT (REFSTRUC) : = NBCAR

. TOTAL (REFSTRUC) := NBCAR

. APRES (REFSTRUC) : = 0

. termine.

REPONSE

REP = 0 si fin normale

REP = - 1 si erreur.

REP = COPVAR (NOM1, REFSTRUC2)

ARGUMENTS D'ACTIVATION

NOM1 : nom de l'entité où l'on copie

REFSTRUC2 = référence du descriptif de l'entité à copier.

OBJET : copie une entité de type variable ou file dans une autre :

. Rend courante l'entité repérée par NOM1 en émettant :

CHZIST (OBLIG, NOM1)

. Vérifie que l'entité courante est une file ou une variable :

si NOMPE (ADCO) ≠ CODFIL ou CODUAR alors :

. émet IMPRIME (INCOMPATIBLE)

. termine.

. Calcule l'adresse du descriptif de l'entité courante :

REFSTRUC1 : = REF (ADCO)

. Copie :

. teste si possible : le nombre maximum de caractères de l'entité 'but' doit être inférieur au nombre total de caractères de l'entité 'source" :

si NBCMAX (REFSTRUC1) TOTAL (REFSTRUC2) alors :

. émet IMPRIME (INCOMPATIBLE)

. termine

. copie l'entité repérée par REFSTRUC2 dans celle repérée par REFSTRUC1

. Met à jour le descriptif de l'entité modifiée :

. TOTAL (REFSTRUC1) : = TOTAL (REFSTRUC2)

. AVANT (REFSTRUC1) : = AVANT (REFSTRUC2)

. APRES (REFSTRUC1) : = APRES (REFSTRUC2)

. COURANT (REFSTRUC1) : = DEBUT (REFSTRUC1) + AVANT (REFSTRUC1)

. FIN (REFSTRUC1) : = COURANT (REFSTRUC1) + APRES (REFSTRUC1)

. Termine.

REPONSE

REP = 0 si fin normale

REP = - 1 si erreur.

REP = CREFIL (NBCENT, REFSTRUC)

OBJET : crée une entité de type FILE et l'initialise à vide :

. alloue un descriptif de file : émet :

ALLOC (LGDFIL, REFSTRUC) si REP = - 1 termine

. calcule la longueur en mots de la chaine pour allocation :

. si la longueur n'a pas été précisée (NBCENT : 0) prend :

NBCENT : = 64 par défaut

. LGENT : = CEIL [NBCENT/4] (4 caractères par mot)

. alloue le contenu : émet :

ALLOC (LGENT, REFCONT) si REP = - 1 termine

. initialise le descriptif :

NBCMAX (REFSTRUC) : = NBCENT

TOTAL (REFSTRUC) : = 0

DEBUT (REFSTRUC) : = REFCONT

FIN (REFSTRUC) : = REFCONT

AVANT (REFSTRUC) : = 0

APRES (REFSTRUC) : = 0

. initialise à vide le contenu : émet :

RAZ (REFCONT, LGENT)

ARGUMENT D'ACTIVATION

NBCENT = longueur de la file en caractères, ou 0 si pas précisée

RESULTAT :

REFSTRUC = référence du descriptif de l'entité

REPONSE

REP = - 1 : allocation impossible : plus de place

REP = 0 si fin normale.

REP = ECRIFIL (REFSTRUC, PTCHAINE)

ARGUMENT D'ACTIVATION : REFSTRUC = référence du descriptif de la file
PTCHAINE = repère la position à partir de laquelle on écrit

OBJET : modifie le contenu d'une file :

. Initialise le descriptif :

. COURANT (REFSTRUC) : = PTCHAINE

. AVANT (REFSTRUC) : = COURANT (REFSTRUC) - DEBUT (REFSTRUC)

. modifie le contenu après courant :

. AVANT : = AVANT (REFSTRUC)

. NBCMAX : = NBCMAX (REFSTRUC)

. émet CHAINE (PTCHAINE, AVANT, NBCMAX, NBCAR)

si REP = - 1 termine

. met à jour le descriptif :

. COURANT (REFSTRUC) : = PTCHAINE

. FIN (REFSTRUC) : = PTCHAINE

. AVANT (REFSTRUC) + = NBCAR

. TOTAL (REFSTRUC) + = NBCAR

. APRES (REFSTRUC) : = 0

. termine.

REPONSE

REP = 0 si fin normale

REP = - 1 si erreur.

REP : POSFIL (REFDESC, REFSTRUC)

ARGUMENT D'ACTIVATION

REFDESC = référence du descriptif de position :

REFSTRUC : référence du descriptif de la file

OBJET : positionne le pointeur courant :

. vérifie les arguments de position : en émettant :

CONTROLE (REFDESC, REFSTRUC, PTCHAINE)

si REP = - 1 termine

. met à jour le descriptif :

. COURANT (REFSTRUC) : = PTCHAINE

. APRES (REFSTRUC) : = FIN (REFSTRUC) - PTCHAINE

. AVANT (REFSTRUC) : = TOTAL (REFSTRUC) - APRES (REFSTRUC)

. termine :

REPONSE :

REP = 0 fin normale

REP = - 1 si erreur.

REP = CONTROLE (REFDESC, REFSTRUC, PTCHAINE)

ARGUMENTS D'ACTIVATION :

REFDESC = référence du descriptif de position

REFSTRUC = référence du descriptif de la file

RESULTAT :

PTCHAINE = pointeur sur position, si correcte.

OBJET : vérifie que les arguments de position sont corrects ; positionne le pointeur PTCHAINE sur la position.

. si RES (REFDESC) : CODEBUT alors :
si SIGN (REFDESC) = - alors : saute en a)
sinon : si INC (REFDESC) > TOTAL (REFSTRUC) alors : saute en a)
sinon : PTCHAINE : = DEBUT (REFSTRUC) + INC

. si RES (REFDESC) = CODCOU alors :
si SIGN (REFDESC) = - alors : si AVANT (REFSTRUC) - INC (REFDESC) < 0
alors : saute en a)
sinon : PTCHAINE = COURANT
(REFSTRUC) - INC
sinon : si AVANT (REFSTRUC) + INC (REFDESC)
> TOTAL (REFSTRUC)
alors : saute en a)
sinon : PTCHAINE = COURANT
(REFSTRUC) + INC

. si RES (REFDESC) = CODFI alors :
si SIGN (REFDESC) = - alors : si TOTAL (REFSTRUC) - INC
(REFSTRUC) < 0
alors : pointe en a)
sinon : PTCHAINE = FIN (REFSTRUC)
- INC
sinon : saute en a)

. termine (REP = 0)

a) . émet IMPRIME (ERREUR POSITION)

. termine (REP = - 1)

REPONSE :

REP = 0 fin normale

REP = - 1 si erreur position.

REP = CHAINE (PTCHAINE, AVANT NBCMAX, NBCAR)

ARGUMENTS D'ACTIVATION :

PTCHAINE = pointeur courant

AVANT = nombre de caractères avant courant

NBCMAX = nombre de caractères maximum de la chaine

RESULTAT :

PTCHAINE = nouveau pointeur courant

NBCAR = nombre de caractères ajoutés.

OBJET : entre une chaîne et la donne en contenu à une entité repérée par PTCHAINE :

a) . Initialisation : NBCAR : = 0

. Entre une ligne TTY en émettant :

ENTLIGN (CARLU) si REP = - 1 alors :

. émet MASCAR (PTCHAINE)

. termine

. Teste si chaine correcte : si AVANT + CARW > NBCMAX alors :

. émet IMPRIME (DDC CHAINE)

. termine

. Ajoute la ligne lue à l'entité :

. PTZONE : = REPONS - 2

. LGZ : = CARLU

. émet ADDCH (PTCHAINE, PTZONE, LGZ, CAREN)

. Met à jour le compteur :

NBCAR : = NBCAR + CAREN

. Retourne en a)

REPONSE :

REP = 0 : fin normale

REP = - 1 : si DDC chaîne.

REP = ADDCH (PTCHAINE, PTZONE, LGZ, CAREN)

ARGUMENT D'ACTIVATION :

PTCHAINE = pointeur dans chaine

PTZONE = pointeur dans zone

LGZ = nombre de caractères à ajouter

RESULTAT :

CAREN = caractères entrés

PTCHAINE = nouveau pointeur dans chaine

OBJET : ajoute à une chaine des caractères contenus dans une zone :

. Initialise CAREN = 0 ; PTZONE pointe sur ZONE (PTZONE) ;

a). tant que LGZ 0 faire :

. si caractère repéré par PTZONE ≠ tous alors :

. place le caractère repéré par PTZONE dans le byte de la zone repérée par PTCHAINE

. incrémente les pointeurs et compteur :

. PTCHAINE + = 1

. PTZONE + = 1

. CAREN + = 1

sinon : incrémente le pointeur dans zone :

PTZONE + = 1

. retourne en a)

. termine.

REPONSE :

REP = 0 fin normale.

REP = REFERENCE (REFDESC1, REFDESC2, REFSTRUC, PTCHAINE, LGCHAINE)

ARGUMENT d'ACTIVATION :

REFDESC1, REFDESC2 : référence des descriptifs de position

REFSTRUC : référence du descriptif de la file

RESULTAT :

PTCHAINE : pointeur sur file

LGCHAINE : longueur de la chaine

OBJET : vérifie la compatibilité des arguments de position ; positionne sur pointeur et calcule la longueur de la chaine :

1). traite la première position :

. Cas RES (REFDESC1) = CODEBUT : PT1 : = DEBUT (REFSTRUC) + INC (REFDESC1)

. Cas RES (REFDESC1) = CODCOU :

. Cas SIGN (REFDESC1) = - :

PT1 : = COURANT (REFSTRUC) - INC (REFDESC1)

. Cas SIGN (REFDES1) = + :

PT1 : = COURANT (REFSTRUC) + INC (REFDESC1)

. Cas RES (REFDESC1) = CODFIN : PT1 : = FIN (REFSTRUC) - INC (REFDESC1)

2). traite la deuxième position :

. Cas RES (REFDESC2) = CODEBUT : PT2 : = DEBUT (REFSTRUC) + INC (REFDESC2)

. Cas RES (REFDESC2) = CODCOU :

. Cas SIGN (REFDESC2) = - :

PT2 : = COURANT (REFSTRUC) - INC (REFDESC2)

. Cas SIGN (REFDESC2) = + :

PT2 : = COURANT (REFSTRUC) + INC (REFSTRUC)

. Cas RES (REFDESC1) = CODFIN : PT2 : = FIN (REFSTRUC) - INC (REFDESC2)

3). Comparaison : si PT1 > PT2 alors :

. émet IMPRIME (POINTEURS INVERSES)

. termine (REP = - 1)

sinon : . PTCHAINE : = PT1

. LGCHAINE : = PT2 - PT1

4). Termine (REP = 0)

REPONSE :

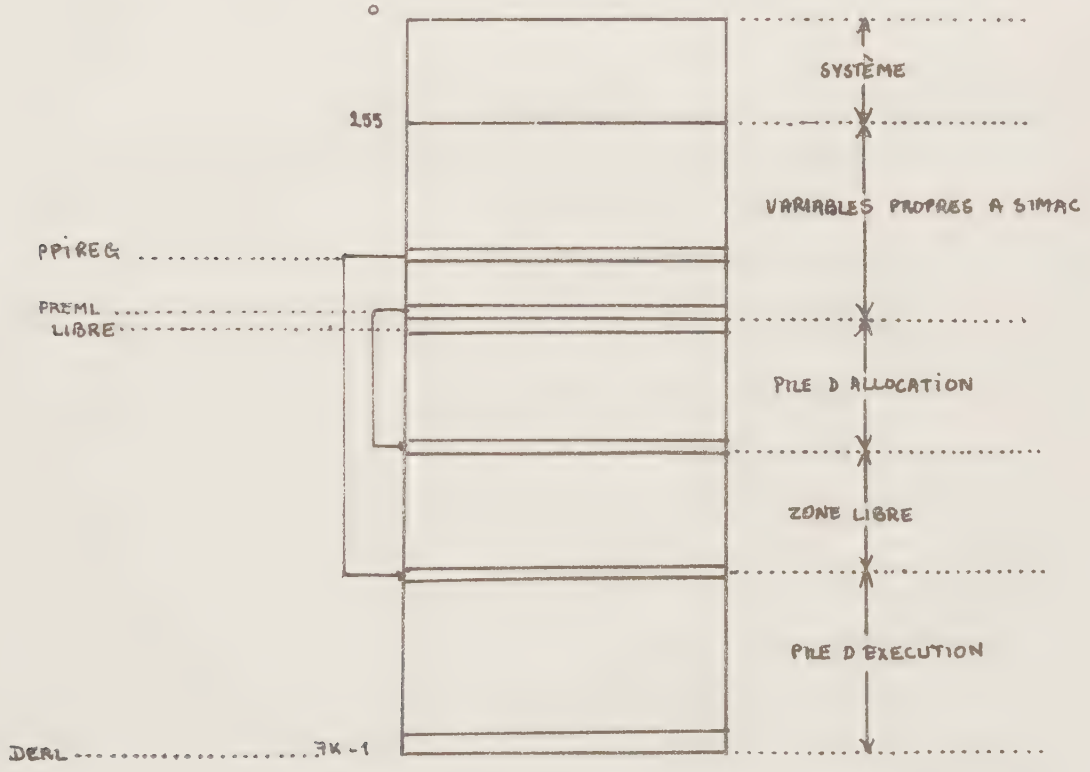
REP = - 1 si erreur

REP = 0 fin normale

2.8. LA MACHINE ALLOCATEUR

OBJET : gère l'espace alloué à l'usager par le système
pour son travail sur SIMAC

STRUCTURE MANIPULEE : la ZONE USAGER se présente comme suit



PRIMITIVES : cf. page suivante :

REP = INITALLOC

Pas d'arguments

OBJET : initialise le pointeur de la pile d'allocation
. PREML : = LIBRE
. termine

Pas de réponse.

REP = ALLOC (LONGUEUR, REFALLOC)

ARGUMENTS D'ACTIVATION :
LONGUEUR = longueur désirée
RESULTAT :
REFALLOC = référence de la zone allouée, si allocation possible
0 sinon.

OBJET : alloue la longueur demandée si possible :
si PREML + LONGUEUR < PPIREG alors :
. REFALLOC : = PREML
. PREML : = PREML + LONGUEUR
. termine
sinon :
. IMPRIME (PLUS DE PLACE)
. REFALLOC : = 0
. termine.

REPONSE :
REP = 0 si fin normale
REP = - 1 si plus de place.

REP = DESALLOC (LONGUEUR)

ARGUMENT D'ACTIVATION
LONGUEUR = longueur à désallouer

OBJET : désalloue la longueur spécifiée
. PREML = PREML - LONGUEUR
. termine

REPONSE :
REP = 0 fin normale.

2.9. LA MACHINE ANALYSE

OBJET : analyse les commandes et les réponses de l'usager

MODULES :

MODULE : ANALYSEUR SYNTAXIQUE

OBJET : reconnait les éléments du vocabulaire des commandes

PRIMITIVES : cf. pages suivantes.

REP = CHTYPE (OBLNOBL, CONTENT, PARAM, TYP)

OBJET : recherche un argument de type entité ou de type contexte

1. recherche de l'argument :
   . TERM : = '=' terminateur voulu
   . émet : REP = DECOD (PARAM, LGTYPE, TERM)

. si l'argument est absent (REP = 0)
   . s'il n'est pas obligatoire (OBLNOBL = NOBLIG) : termine
   . s'il est obligatoire (OBLNOBL = OBLIG) :
      2. demande le type : émet :
         LECT (TYPE)
         . si REP = 0 : termine
         . si REP 0 : retourne en 1

. si l'argument est présent (REP 0)
   . s'il est de type contexte (CONTENT = CONTEXTE) recherche en table :
      TABLE (PARAM, TCONT, REFITEM)
      . si REP = - 1 : retourne en 2
      . si REP ⩾ 0 : TYP : = REFITEM
   . s'il est de type entité (CONTENT = ENTITE) recherche en table :
      TABLE (PARAM, TENT, REFITEM)
      . si REP = - 1 : retourne en 2
      . si REP > 0 : TYP : = REFITEM
   . termine.

ARGUMENT D'ACTIVATION :

OBLNOBL = OBLIG si type obligatoire , = NOBLIG sinon
CONTENT = CONTEXTE si type contexte , = ENTITE si type entité

RESULTAT :

PARAM = type si trouvé
TYP = n° de type si trouvé

REPONSE :

REP = - 1 si argument non présent et non obligatoire
REP = - 2 si argument non présent et obligatoire
REP ⩾ 0 si argument présent.

REP = CHNOM (OBLNOBL, ZNOM)

OBJET : recherche un argument NOM

1. recherche de l'argument :
   . TERM : = '(' terminateur voulu
   . émet : REP = DECOD (ZNOM, LGNOM, TERM)

. si le nom est absent : (REP = 0)
   . s'il n'est pas obligatoire (OBLNOBL = NOBLIG) : termine
   . s'il est obligatoire (OBLNOBL = OBLIG) :
      . demande le nom : émet :
         LECT (NOM :)
         . si REP = 0 : termine
         . si REP 0 : retourne en 1

. si le nom est présent : (REP 0) termine.

ARGUMENT : D'ACTIVATION :

OBLNOBL = OBLIG si type obligatoire, = NOBLIG sinon

RESULTAT :

ZNOM = nom si trouvé

REPONSE :

REP = - 1 si nom non présent et non obligatoire

REP = - 2 si nom non présent et obligatoire

REP 0 si nom présent.

REP = CHENTI (OBLNOBL, NBCENT)

OBJET : recherche un argument entier, si celui-ci est présent le convertit en binaire.

1. recherche un argument entier :
   . TERM : = ')'
   . émet : DECOD (NBCENT, LGENTI, TERM)
. si l'argument est absent (REP = 0) :
   . s'il n'est pas obligatoire (OBLNOBL = NOBLIG) :
      . NBCENT : = 0
      . termine
   . s'il est obligatoire (OBLNOBL = OBLIG) :
      2. demande l'argument : émet :
            LECT (ENTIER)
               . si REP = 0 termine
               . si REP > 0 retourne en 1
. si l'argument est présent (REP > 0)
   . vérifie qu'il n'est pas précédé du signe moins :
      . si précédé du signe moins retourne en 2
   . effectue la conversion : émet :
      NOMBRE (NBCENT, LGENTI)
            . si REP = - 1 : retourne en 2
            . si REP = 0 termine

ARGUMENTS : D'ACTIVATION

OBNOBL = OBLIG si obligatoire, NOBLIG sinon
   RESULTAT
NBCENT = entier trouvé en binaire, ou 0 si pas d'entier

REPONSE

REP = - 2 si argument non présent quand demandé
REP = - 1 si argument non présent et non obligatoire
REP = 0 si argument présent et correct.

REP = CHPOS (REFDESC, TERM)

ARGUMENT D'ACTIVATION

REFDESC = référence du descriptif de position à remplir

RESULTAT

TERM = terminateur trouvé.

OBJET : recherche un argument de type POSITION :

position : = < pointeur > | < pointeur > $\overset{+}{-}$ < inc >

1) traite le pointeur :

. recherche le pointeur :

. TERM : = - 1

. émet DECOD (PARAM, LGPOINT, TERM)

si REP = 0 alors termine (REP = 0)

. vérifie la validité du pointeur en émettant :

TABLE (PARAM, TPOINT, REFITEM)

si REP = - 1 alors termine (REP = - 1)

. place le N° de pointeur dans le descriptif :

RES (REFDESC) : = REFITEM

2) traite le signe :

. si TERM = , alors termine (REP > 0)

. si TERM = + alors SIGN (REFDESC) : = TERM

. si TERM = - alors SIGN (REFDESC) : = TERM

. si autre terminateur alors termine (REP = - 1)

3) traite l'incrément :

. recherche l'incrément :

. TERM : = - 1

. émet DECOD (PARAM, LGINC, TERM)

si REP = 0 termine (REP = - 1)

. convertit le nombre en binaire en émettant :

NOMBRE (PARAM, LGINC)

si REP = - 1 termine (REP = - 1)

. place le nombre binaire dans le descriptif de position :

INC (REFDESC) : = PARAM

4) termine. (REP > 0)

REPONSE

REP = 0 argument absent

REP = - 1 si erreur

REP > 0 si fin normale.

MODULE : FONCTIONS DE LECTURE

OBJET : gère le tampon REPONS en entrée ;

STRUCTURE MANIPULEE : le tampon REPONS (cf. visualisateur).

REP = CARTAMP (CAR)

ARGUMENT RESULTAT

CAR = cellule où l'on place le caractère.

OBJET : place le caractère repéré par le pointeur du tampon d'E/S dans la cellule
CAR, ignore les blancs :

. si REPONS [0] = 0 alors termine

a). met le caractère repéré par PTREP dans CAR

. si CAR = blanc alors : . PTREP + = 1

. REPONS [0] = REPONS [0] - 1

. retourne en a

sinon : . PTREP + = 1

. REPONS [0] = REPONS [0] - 1

. termine

REPONSE :

REP = 0 fin normale

REP = - 1 fin de ligne.

REP = DECOD (REFTAMP, LGARG, TERM)

OBJET : prend un argument dans le buffer REPONS et le met dans un TAMPON, rend le TERMINATEUR dans le cas où il est demandé.

ARGUMENTS : D'ACTIVATION

REFTAMP = référence du tampon où on écrira l'argument
LGARG = longueur maximum de l'argument
TERM = caractère terminateur, = - 1 si terminateur désiré au retour
RESULTAT
TERM = terminateur si voulu, 0 sinon
REFTAMP = référence le tampon où figure l'argument

REPONSE :

REP = 0 pas d'argument dans le buffer
REP > 0 si fin normale.

REP = OUINON (MESSAGE, REPONSE)

OBJET : sort un message et attend OUI ou NON
recommence si la réponse est différente de OUI ou NON
termine si réponse vide

ARGUMENT D'ACTIVATION.

MESSAGE = message à sortir
RESULTAT :
REPONSE = OUI ou NON

REPONSE :

REP = 0 si fin normale
REP = - 1 si réponse vide.

REP = ENTLIGN (CARLU)

ARGUMENT RESULTAT

CARLU = nombre de caractères lus

OBJET : entre une ligne TTY et la place dans le tampon d'E/S

REPONSE

REP = - 1 ligne vide

REP = 0 ligne non vide.

REP = LECT (MESSAGE)

OBJET : sort un message et place la réponse de l'usager dans le buffer REPONS

ARGUMENT : D'ACTIVATION

MESSAGE = message à sortir

REPONSE :

REP = 0 si réponse vide

REP > 0 si réponse non vide.

2.10.LA MACHINE VISUALISATEUR

OBJET : permet la visualisation des entités créées par l'usager, ainsi que l'édition des diagnostics

MODULES :

MODULE : DEMANDES DE VISUALISATION

OBJET : réalise les éditions demandées soit par l'usager, soit par l'exécuteur.

PRIMITIVES :

REP = LISTER

Pas d'arguments

OBJET : liste le nom de toutes les entités de type donné, liste toutes les entités suivies de leur type si le type n'est pas donné ; traite la commande

COM : LISTER : < TYPE >

1) . analyse la commande :
recherche un type d'entité non obligatoire en émettant :
CHTYPE (NOBLIG, ENTITE, PARAM, TYP)

2) . traite la commande :
. Cas REP = - 1 : type pas précisé :
édite le nom et le type de toutes les entités
en émettant :
EDNOTYP
. Cas REP = 0 : type précisé : édite le nom de toutes les entités du dype donné, en émettant :
EDNO (TYP)

3) . termine

REPONSE

REP = 0 si fin normale.

REP = REGLES

Pas d'arguments

OBJET : édite la pile des règles appliquées, constituée lors de la dernière activation en traitant la commande : COM : REGLES

. Teste la pile d'execution : émet :
PILDIV (REPONSE)
si REPONSE = OUI alors :
. émet IMPRIME (PILE VIDE)
. termine

. Edite les règles :
. émet IMPRIME (REGLES UTILISEES)
. initialise avant boucle : POINTEUR : = DERL
1). boucle :
. sortie d'une règle : émet : PEDIR (POINTEUR, COAUTO)
. teste si fin : émet : PPIREG (POINTEUR, REPONSE)
si REPONSE = NON retourne en 1)

. Termine

REPONSE

REP = 0 fin normale.

REP = EDITER

Pas d'argument

OBJET : édite l'entité demandée en traitant la commande :

EDITER : < NOM >, < REFERENCE > | < PARAMETRE >

< PARAMETRE > = REGLE | FORME

< REFERENCE > = < POSITION > , < POSITION > |, < POSITION > |
< POSITION > | Ø

1) traite la commande :
   . recherche un nom, non obligatoire :
     CHNOM (NOBLIG, ZNOM)
       si REP = - 1 . émet EDITCONTEX
                    . termine (REP = 0)
2) traite le NOM, recherche si un identificateur porte déjà ce nom : en émettant : CHLIST (ZNOM)
       si REP = - 1 . émet IMPRIME (ENTITE INCONNUE)
                    . termine (REP = -1)
3) édite l'entité courante : émet :
     EDITENT si REP = -1 termine (REP = -1)
4) termine

REPONSE

REP = 0 si fin normale

REP = - 1 si erreur.

REP = IMPENT (MESSAGE, REFSTRUC, MODE)

ARGUMENT D'ACTIVATION :

MESSAGE = message à imprimer avant le contenu

REFSTRUC = référence du descriptif de l'entité à

MODE = 0 si on imprime TOUT

2 si on imprime de COURANT à FIN

4 si on imprime en 2 fois

OBJET : Imprime le contenu d'une entité :

. prépare 1'impression :

. Cas MODE = 0 : . PTSORT : = DEBUT (REFSTRUC)
. LGSORT : = TOTAL (REFSTRUC)
. émet IMPRIME (MESSAGE

. Cas MODE = 2 : . PTSORT : = COURANT (REFSTRUC)
. LGSORT : = APRES (REFSTRUC)
. émet : IMPRIME (MESSAGE)

. Cas MODE = 4 : . PTSORT : = DEBUT (REFSTRUC)
. LGSORT : = AVANT (REFSTRUC)
. émet : IMPRIME (MESSAGE)
. émet : IMPRIME (DEBUT :)

. impression :

. émet : IMPCH (PTSORT, LGSORT)

. si MODE = 4 alors . émet : IMPRIME (FIN :)
. LGSORT : = APRES (REFSTRUC)
. PTSORT : = COURANT (REFSTRUC)
. émet IMPCH (PTSORT, LGSORT)

. termine

REPONSE

REP = 0 si fin normale.

REP = IMPCH (PTSORT, LGSORT)

ARGUMENTS D'ACTIVATION :

PTSORT = pointeur sur chaîne à sortir

LGSORT = longueur de la chaîne

OBJET : sortie d'une chaîne de caractère :

. si LGSORT = 0 alors : émet IMPRIME (VIDE) et termine

. place la chaîne dans le tampon REPONS :

EMPLITAMP (PTSORT, LGSORT)

. REPONS [0] : = LGSORT

. sort la chaîne : émet EDITAMP

. termine

REPONSE

REP = 0 si fin normale.

REP = PEDIR (SAUV, COAUTO)

ARGUMENTS D'ACTIVATION :

SAUV : pointeur sur règle à éditer

COAUTO : référence de l'automate courant

RESULTAT :

SAUV : pointeur sur premier champ de la règle suivante

OBJET : édition d'une règle exécutée :

. constitution de la pile des séparateurs de champs : émet :
SEPAR (COAUTO, SEP, SAVEM, NBCHAMP)

. initialise la boucle :

. NCP pointe sur SEP [0]

. PTSORT : = SAUV (pointe sur 1er champ)

. NBCAR : = NBCHAMP

1). boucle :

. mise en place du champ dans le tampon REPONS

. mise en place du séparateur dans le tampon REPONS

. NBCAR - = 1

. si NBCAR 0 : alors :

. incrémente :

. PTSORT - = 1

. NCP - = 1

. retourne en 1)

sinon : sort le tampon : émet :
EDITAMP

. progresse dans la pile d'exécution : SAUV + = NBCHAMP

. termine

REPONSE

REP = 0 fin normale.

REP = SEPAR (COAUTO, SEP, SAVEM, NBCHAMP)

ARGUMENT D'ACTIVATION :

COAUTO = référence de l'automate courant

RESULTAT :

SEP = pile des séparateurs

SAVEM = nombre des champs gauches

NBCHAMP = nombre de champs total

OBJET : constitution de la pile des opérateurs avant édition d'une règle :

. séparateur entre champs du même membre = ','

. séparateur entre membres ' —>'

. séparateur suivant le dernier champ de la règle : '.'

. termine

REPONSE

REP = 0 fin normale.

MODULE : FONCTIONS DE VISUALISATION :

OBJET : gère le tampon REPONS en sortie ;

STRUCTURE MANIPULEE : le tampon REPONS : il contient cadrés à gauche les caractères à éditer.

PRIMITIVES :

REP = INITAMP

Pas d'argument

OBJET : initialise le tampon d'entrée - sortie :

- . met à zéro les cellules du tampon : en émettant : RAZ (REPONS, LGREPONS)
- . initialise le pointeur PTREP en début de tampon : PTREP pointe sur REPONS [1]
- . termine

REPONSE

REP = 0 : fin normale.

REP = EMPLITAMP (REFITEM, LGITEM)

ARGUMENT D'ACTIVATION

REFITEM = référence de l'élément à placer dans le tampon

LGITEM = longueur de l'élément en caractères

OBJET : remplit le tampon d'entrée - sortie REPONS :

. place l'élément référencé par son adresse dans le tampon

. met à jour le pointeur PTREP :

PTREP : = PTREP + LGITEM

. met à jour le compteur REPONS :

REPONS [0]: = REPONS [0] + LGITEM

. termine

REPONSE

REP = 0 : fin normale

REP = EDITAMP

Pas d'argument

OBJET : . sortie sur TTY du contenu du tampon d'entrée - sortie

. termine

REPONSE

REP = 0 fin normale.

REP = IMPRIME (MESSAGE)

ARGUMENT D'ACTIVATION

MESSAGE = chaîne de caractère à sortir

OBJET : sortie sur TTY du message : MESSAGE

REPONSE

REP = 0 fin normale.

V-2.11.LA MACHINE UTILITAIRES

OBJET : regroupe les programmes utilitaires utilisés dans SIMAC

PRIMITIVES :

REP = NOMBRE (ENTIER, LGENTI)

OBJET : convertit en binaire un entier signé

ARGUMENTS : D'ACTIVATION
ENTIER = entier signé en caractères
LGENTI = longueur en caractères de l'entier
RESULTAT
ENTIER = entier signé en binaire

REPONSE :
REP = - 1 si l'argument n'est pas un entier
REP = 0 si fin normale.

REP = RAZ (POINTEUR, LONGUEUR)

OBJET : remet à zéro la zone repérée par POINTEUR et LONGUEUR

ARGUMENTS D'ACTIVATION :
POINTEUR pointe sur le premier mot de la zone
LONGUEUR = longueur en mots de la zone

REPONSE :
REP = 0 fin normale.

REP = TABLE (PARAM, REFTABLE, REFITEM)

OBJET : recherche d'un item dans une table

ARGUMENTS : D'ACTIVATION :

PARAM = item à rechercher

REFTABLE = référence de la table

RESULTAT :

REFITEM = référence de l'item dans la table si trouvé,
- 1 sinon

REPONSE :

REP = - 1 si pas d'item ou table vide

REP > 0 si item trouvé

TABLES MANIPULEES :

TENT = table des types d'entités

TCONT = table des types de contextes

TOPCO = table des opérateurs courants

TPOINT = table des pointeurs

TPAR = table des paramètres.

CHAPITRE VI

CONCLUSION

L'apport de SIMAC 1.5, relativement aux autres versions, se situe au niveau des facilités d'accès au système offertes au programmeur.

La conception originelle de SIMAC offre la souplesse et la maniabilité nécessaires pour aider efficacement l'enseignement et la recherche en déchargeant l'utilisateur de toute question de mise en oeuvre et en lui permettant de se consacrer entièrement à la description des automates et à leur mise au point.

Notre travail permet d'enrichir le système en offrant de grandes facilités d'y introduire des modifications souhaitées ou de nouveaux objets sans trop d'effort de programmation et sans supprimer la transparence du système pour l'usager.

L'accès au système se trouve facilité par la description obtenue par analyse descendante : la structure d'arborescence du programme, sa hiérarchisation en niveaux, permettent de suivre facilement le chemin qui conduit d'une commande de l'usager jusqu'au traitement des objets concernés par cette commande. De plus, les modules composant la description, étant indépendants, peuvent être compris indépendamment de leur contexte d'utilisation. Enfin, la paramétrisation, en associant un nom à chacun des paramètres du programme et en explicitant ce nom permet une compréhension immédiate du rôle de chaque valeur.

Ainsi :

- la sémantique de chaque objet étant clairement définie (spécification),
- les différents traitements de ces objets n'interférant pas (modularité),
- l'enchaînement des traitements étant parfaitement décrit (structuration),

le programmeur peut donc :

- accéder rapidement et sans gros effort à la compréhension du système,
- disposer de toutes les possibilités offertes par cette conception du système.

Les possibilités de cette version peuvent être regroupées selon trois rubriques : possibilité de transplantation, de modification, d'évolution.

Au niveau des possibilités de transplantation, l'utilisation d'un langage spécifique et la paramétrisation du programme permettent de se dégager complètement de l'implantation actuelle de SIMAC, du langage d'assemblage et des contraintes de l'ordinateur utilisé. Ceci doit faciliter l'implantation du système sur une machine quelconque dans un langage au choix : ce qui est très important étant donné la vocation pédagogique de SIMAC.

Les modifications voulues par le concepteur n'auront à être effectuées qu'à un seul endroit du programme du fait de sa structure d'arborescence et de la non-interférence des différents modules. Par exemple : le principe général de l'allocation peut être modifié ou adapté au gré du programmeur sans pour autant modifier l'ensemble des primitives.

Les évolutions possibles de SIMAC 1.5 sont de deux types :
- définition de nouvelles classes de machines,
- introduction de nouveaux modes d'exécution des machines.

De nouvelles classes de machines peuvent être définies par introduction de nouvelles entités et de nouveaux opérateurs chargés de les manipuler lors de l'activation. Le travail se limite alors aux opérations suivantes :

- création d'un module de gestion de la nouvelle entité : c'est-à-dire écriture des primitives de création, manipulation, destruction de cette entité ;
- définition d'opérateurs sur cette entité : c'est-à-dire écriture pour chaque fonction d'exécution, d'une procédure spécifiant le travail effectué sur l'entité lors de l'appel de cette fonction à l'exécution.

De nouveaux types d'exécuteurs peuvent être également définis ; c'est alors le module Activation de la machine exécution qui s'enrichit d'un nouvel exécuteur.

Le nouveau mode d'exécution pourra nécessiter de nouvelles fonctions d'exécution, celles-ci nécessiteront à leur tour l'écriture des procédures, éventuellement vides, associées à chaque opérateur.

Dans le cas le plus général, un nouvel exécuteur pourra exiger un archivage des règles différent de celui employé jusqu'ici : c'est alors le module de gestion des règles qui s'agrandira. On remarque que puisque plusieurs options d'archivage seront alors offertes, l'usager devra préciser au moment de la définition d'un automate l'option d'archivage désirée.

Ainsi, il apparaît qu'une évolution de SIMAC comporte toujours deux parties :
- une partie traitant l'archivage : par exemple implantation d'une nouvelle entité, nouvel archivage des règles, ...
- une partie traitant l'activation : par exemple nouvel opérateur, nouvelle recherche,...

La structuration du système permet dès à présent de localiser les parties de programme concernées par telle ou telle évolution :
- nouvelle entité, nouvel archivage des règles : module CONTENU de la machine ARCHIVAGE,
- nouvel opérateur : machine OPERATEUR,
- nouvel exécuteur : module ACTIVATION de la machine EXECUTION.

Enfin, la modularité garantit que seuls ces modules ainsi délimités sont concernés.

Finalement SIMAC 1.5 est devenu un outil pratique non seulement parce qu'il permet la mise en oeuvre d'une grande variété d'automates, mais aussi parce qu'il présente un modèle de machine idéale. Ce modèle, schéma utile de définition de machines complètement nouvelles, permet de compléter sans effort le système en vue de la mise en oeuvre de ces nouvelles machines. Il a de plus aidé à dégager les concepts fondamentaux des machines idéales ce qui est une première voie pour atteindre une variabilité systématique et ainsi définir des machines aussi variées que possible.

Tout au long de notre travail, nous avons constamment cherché à idéaliser les concepts et cet effort s'est avéré particulièrement utile pour le développement d'un nouveau système extensible SIMAC 2. qui est maintenant en cours de réalisation. [DAQ3]

ANNEXE 1

BIBLIOGRAPHIE

ARS — J. ARSAC : De la programmation (Publication iP N° 72.13 Oct. 72)

CHE — R.J. CHEVANCE : Notes du cours de compilation (i.P.72)

DAH — O.J. DAHL - E.W. DIJKSTRA - C.A.R. HOARE : Structured Programming (New-York, London, Academic Press, 1972)

DAQ1 — C. DAQUIN - B. JOURDAN DE LA PASSARDIERE : Simac, une application conversationnelle d'automates (Polycopié i.P. 72)

DAQ2 — C. DAQUIN - C. GIRAULT : Simac, simulateur conversationnel d'automates (RAIRO, B3, Septembre 72)

COU — G. COUSINEAU : Notice d'utilisation de SIMAC (i.P. Juin 72)

COUR — D. COURTOIS - C. DAQUIN - M. DUPRAS : WISP, une application conversationnelle de listes (Polycopié i.P. 72)

DEN 1 — J.B. DENNIS : MODULARITY . (Advanced course on software engineering Munich June 72)

DEN 2 — J.B. DENNIS : The Design and construction of software systems (Advanced course on software engineering. Munich June 72)

DIJ 1 — E.W. DIJKSTRA : Notes on structured programming (University of Maryland. EWD 249. Aug. 69)

DIJ 2 — E.W. DIJKSTRA : A constructive approach tho the problem of program Correctness (BIT, vol.8, N° 3, 1968)

FEL — J. FELDMANN, D. GRIES : Translator writing systems (Comm. ACM. February 1968)

GIN — S. GINSBURG : The Mathematical theory of contest free languages (Mc Graw-hill New-York 1966)

GIR 1 — C. GIRAULT : un des systèmes de multiprogrammation réalisés à l'institut de programmation de la Faculté des Sciences de Paris (RIRO, R2, 1971)

../..

GIR 2 C. GIRAULT : un interpréteur conversationnel d'automates (Journées d'Informatique Théorique de Strasbourg - Février 1971)

GIR 3 C. GIRAULT : Traitement des données (Polycopiés i.P.72)

GOO G. GOOS : Hiérarchies (Advanced course of software engineering Munich June 72)

GRO M. GROSS - A. LENTIN : Notions sur les grammaires formelles (Gauthier-Villars, 1967)

HOP J.E. HOPCROFT - J.D. ULLMAN : Formel languages and their relations to Automate (Addison - Wesley 1969)

LIS B.H. LISKOV - S. ZILLES : Programming with abstract data types (very high level languages symposium Santa Monica Mar.74)

MIL 1 H.D. MILLS : Top-down programming in large system (in debugging techniques in large system. Rustin Ed. Prentice Hall.1971).

MIL 2 H.D. MILLS : Mathematical foundations for structured programming (IBM February 1972).

POU L. POUZIN : Notes de cours (Polycopié I.P. 73)

SIM 1 J.C. SIMON : Représentation et traitement de certaines structures d'information (Polycopié I.P. 73)

SIM 2 J.C. SIMON : Techniques de l'analyse syntaxique (Polycopié I.P. 73)

WIL M.V. WILKES : An experiment with a self compiling compiler for a simple list processing language (Annual review in automatic programming, vol. 4, 1964)

WIR N. WIRTH : Program development by stepwise refinement (CACM Apr. 71)

ZUR F.W. ZURCHER, B. RANDELL : Iterative multilevel modelling. A methodology for computer system design (IFIPS conference Edinburgh 1968).

ACADÉMIE DE PARIS

UNIVERSITÉ DE PARIS VI

4, Place Jussieu - 75230 PARIS-CEDEX 05

SERVICE DE LA SCOLARITE

Bureau des Enseignements du 3e cycle

TÉLÉPHONE : 336-25-25 - 325-12-21

UNIVERSITÉ
PIERRE ET MARIE CURIE

PERMIS D'IMPRIMER

DOCTORAT : D'UNIVERSITE

THESE DE :

NOM : Mademoiselle NIVELT

Prénoms : Martine

né le 7 Février 19 47 à Boulogne -Billancourt Dépt :

1ère THESE : " Structuration et applications d'un simulateur d'automat

2e THESE (Propositions faites par l'Université) :

." Simulation d'équilibres écologiques. "

ESIDENT.M.BERSTEL

AMINATEURS.MM.GIRAULT
ROBINET
LAFARGUE

UNIVERSITÉ PIERRE ET MARIE CURIE PARIS 6

Vu et Approuvé

PARIS, le 11 Mars 1975

Le Président

de l'Université de PARIS VI

ACADÉMIE DE PARIS

UNIVERSITÉ DE PARIS VI

4, Place Jussieu - 75230 PARIS-CEDEX 05

SERVICE DE LA SCOLARITE

Bureau des Enseignements du 3e cycle

TÉLÉPHONE : 336-25-25 - 326-12-21

UNIVERSITÉ
PIERRE ET MARIE CURIE

PERMIS D'IMPRIMER

DOCTORAT : D'UNIVERSITE

THESE DE :

NOM : Monsieur GOY

Prénoms : Joël

né le 2 Décembre 1950 à Asnières Dépt : de la Sei

1ère THESE : " Structuration et applications d'un simulateur d'automat

2e THESE (Propositions faites par l'Université) :

" Méthodes d'identification de machines. "

DENT.M.BERSTEL

INATEURS.MM.GIRAULT

ROBINET

UNIVERSITÉ PIERRE et MARIE CURIE PARIS 6

Vu et Approuvé

PARIS, le 11 Mars 1975

Le Président

de l'Université de PARIS VI